Journal of Global Optimization (2025) 92:55–85
https://doi.org/10.1007/s10898-025-01481-w

Limited memory bundle DC algorithm for sparse pairwise kernel learning

Napsu Karmitsa[1] · **Kaisa Joki**[2] · **Antti Airola**[1] · **Tapio Pahikkala**[1]

Received: 30 October 2023 / Accepted: 11 March 2025 / Published online: 3 April 2025

Abstract
Pairwise learning is a specialized form of supervised learning that focuses on predicting outcomes for pairs of objects. In this paper, we formulate the pairwise learning problem as a difference of convex (DC) optimization problem using the Kronecker product kernel, ℓ_1- and ℓ_0-regularizations, and various, possibly nonsmooth, loss functions. Our aim is to develop an efficient learning algorithm, SPARSEPKL, that produces accurate predictions with the desired sparsity level. In addition, we propose a novel limited memory bundle DC algorithm (LMB-DCA) for large-scale nonsmooth DC optimization and apply it as an underlying solver in the SPARSEPKL. The performance of the SPARSEPKL-algorithm is studied in seven real-world drug-target interaction data and the results are compared with those of the state-of-art methods in pairwise learning.

Keywords Pairwise kernel learning · Nonsmooth DC optimization · Bundle methods · DCA · ℓ_0-pseudo-norm · Zero-shot learning

1 Introduction

Pairwise learning corresponds to the supervised learning setting with the aim of making predictions for pairs of objects. Data with pairwise observations naturally arise, for instance, in recommender systems [20, 31, 42], information retrieval [29], *drug-target interaction* (DTI) prediction [6, 39], and link prediction in social networks [47, 49] to mention but a few. In this paper, we formulate the pairwise learning problem as a *difference of convex* (DC) optimization problem using the Kronecker product kernel, ℓ_1- and ℓ_0-regularizations, and various, possible nonsmooth, loss functions. Our aim is to create a learning algorithm—SPARSEPKL —that

✉ Napsu Karmitsa
napsu@karmitsa.fi

Kaisa Joki
kjjoki@utu.fi

Antti Airola
ajairo@utu.fi

Tapio Pahikkala
aatapa@utu.fi

[1] Department of Computing, University of Turku, 20014 Turku, Finland

[2] Department of Mathematics and Statistics, University of Turku, 20014 Turku, Finland

produces accurate predictions with the desired sparsity level. In particular, we aim to solve the realistic form of the DTI problem called *zero-shot learning*. It corresponds to predicting labels for such drug-target pairs where neither the drug nor the target is encountered during the training phase.

In addition, we propose a novel *limited memory bundle DC algorithm* (LMB- DCA) for large-scale nonsmooth DC optimization and apply it as an underlying solver in the SPARSEPKL algorithm. We would like to emphasize that the proposed LMB- DCA is not the well-known DCA [27, 41] with the *limited memory bundle method* (LMBM) [15, 16] as an underlying solver but rather the LMBM modified to solve DC optimization problems using the DCA as a backup.

Using a learning algorithm that produces accurate predictions with a sparse model offers benefits such as interpretability, efficiency, better generalization to new data, scalability, and robustness to noise. These advantages make sparse models desirable in various applications (see, e.g., [10, 11, 18, 24, 50, 54]), allowing for improved understanding, computational efficiency, and reliable predictions.

In practice, we tackle the problem of finding sparse solutions by adding an ℓ_0-pseudo-norm as a penalty for the pairwise kernel learning problem. Note that basically the ℓ_0-pseudo-norm simply counts the number of nonzero components of a vector, and the optimization problems involving this pseudo-norm as it is are known to be NP-hard. Therefore, we instead use the continuous formulation of the ℓ_0-pseudo-norm [11, 13] based on the polyhedral k-norms. This allows us to control the sparsity of the solution vector. It is worth of noting that the resulting problem is a nonsmooth DC optimization problem.

Applying the ℓ_0-pseudo-norm in sparse optimization is not a new idea. However, to our knowledge, the continuous formulation of the ℓ_0-pseudo-norm has been applied to a regression problem utilizing kernel transformations only in [37], where the LMBM-Kronℓ_0LS algorithm for sparse pairwise kernel learning problems is introduced. The differences between the SPARSEPKL algorithm proposed here and the LMBM-Kronℓ_0LS algorithm introduced in [37] are that the SPARSEPKL algorithm

- utilizes the exact DC formulation of the sparse pairwise kernel learning problem and solves it with the novel LMB- DCA;
- applies a double regularization scheme instead of the sole ℓ_0-pseudo-norm (see, Sect. 5.1); and
- solves the pairwise kernel learning problem with the user-specified sparsity percentage and with the selected loss function such as the squared loss, squared epsilon-intensive loss, epsilon-intensive squared loss, epsilon-intensive absolute loss, and absolute loss.

In contrast to the last point, the LMBM-Kronℓ_0LS algorithm applies only the squared loss and aims to find a solution as sparse as possible while maintaining satisfactory prediction accuracy with respect to the non-sparse reference solution. Although, finding a solution as sparse as possible may sound like a good idea, it also makes the LMBM-Kronℓ_0LS algorithm rather complex and time-consuming.

The contribution of the paper is three-fold:

1. to introduce a novel algorithm LMB- DCA for large-scale nonsmooth DC optimization;
2. to present a novel algorithm SPARSEPKL to solve sparse pairwise kernel learning problems; and
3. to compare different, possible nonsmooth, formulations of the loss functions when seeking sparse solutions.

The paper is divided into two main parts. The first part considers nonsmooth and DC optimization (Sects. 2 and 3) whereas the second part is devoted to pairwise kernel learn-

ing problems (Sects. 4–7). More precisely, we first give some preliminaries to nonsmooth, DC, and sparse optimization in Sect. 2 and then we propose a new LMB- DCA for solving large-scale nonsmooth DC optimization problems in Sect. 3. In Sect. 4, we present the basics of pairwise learning problems and kernel methods including Kronecker product kernels. Section 5 focuses on the model formulation by recalling the used loss functions with their (sub)gradients and introducing the double regularization procedure used. Further, in Sect. 6 we propose a new SPARSEPKL algorithm for solving pairwise kernel learning problems with the desired sparsity of the solution. This algorithm applies the LMB- DCA as an underlying solver at each iteration. In Sect. 7, we give numerical results for the new SPARSEPKL algorithm and compare the obtained results against the state-of-the-art methods Kronecker product kernel regularized least squares (KronRLS) and Kronecker support vector machine (KronSVM) [1, 38] and against the LMBM-Kronℓ_0LS algorithm [37]. Further, we evaluate the LMB- DCA as a standalone optimization method by comparing it with the well-known DCA. Finally, Sect. 8 concludes the paper. The notations used throughout the paper are listed in Table 1.

2 Preliminaries

We start with a short introduction to nonsmooth and DC optimization. An interested reader can find more details, for example, from [3, 8, 19, 28, 36, 41, 43]. In addition, we recall the basic idea of sparse optimization and define the continuous formulation of the ℓ_0-pseudo-norm.

In what follows, we denote by $\mathbb{R}^n$ the n-dimensional Euclidean space, by $a^\top b = \sum_{i=1}^n a_i b_i$ the inner product of vectors $a, b \in \mathbb{R}^n$ and by $B(a, \varepsilon)$ the open ball at $a \in \mathbb{R}^n$ with a radius $\varepsilon > 0$. The associated ℓ_2- and ℓ_1-norms are denoted by $\|a\|_2 = (a^\top a)^{1/2}$ and $\|a\|_1 = \sum_{i=1}^n |a_i|$, respectively.

2.1 Nonsmooth optimization

A function $J : \mathbb{R}^n \to \mathbb{R}$ is *nonsmooth* if there are one or more points $a \in \mathbb{R}^n$ such that J fails to be differentiable at these points. Typically these points are encountered at extremes of nonsmooth functions [3]. In nonsmooth optimization, we try to find a minimizer (or a maximizer) of the problem, where at least one of the objective functions involved is nonsmooth.

A function $J : \mathbb{R}^n \to \mathbb{R}$ is *convex* if for all $a, b \in \mathbb{R}^n$ and $\lambda \in [0, 1]$ we have

$$J(\lambda a + (1 - \lambda)b) \leq \lambda J(a) + (1 - \lambda) J(b).$$

Further, a function $J : \mathbb{R}^n \to \mathbb{R}$ is *locally Lipschitz continuous* if at every $a \in \mathbb{R}^n$ there exists $L > 0$ and $\varepsilon > 0$ such that

$$|J(b) - J(c)| \leq L\|b - c\|_2 \quad \text{for all } b, c \in B(a, \varepsilon).$$

For example, both convex and smooth functions always satisfy this property. Whenever a function $J : \mathbb{R}^n \to \mathbb{R}$ is locally Lipschitz continuous, the *Clarke subdifferential* at a point $a \in \mathbb{R}^n$ can be calculated with the formula [3, 8]

$$\partial J(a) = \text{conv} \left\{ \lim_{i \to \infty} \nabla J(a^i) \mid a^i \to a \text{ and } \nabla J(a^i) \text{ exists} \right\},$$

Table 1 Notations

$\mathcal{H}$	Hypothesis space (e.g. reproducing kernel Hilbert space)
$\mathcal{D}$, $\mathcal{T}$	Spaces of drugs and targets
$\mathcal{X}$	Input space, i.e., space of drug-target pairs
$\mathcal{Y}$	Label space
$d \in \mathcal{D}$, $t \in \mathcal{T}$	Drugs and targets
X_d, X_t	Sets of observed drugs and targets, $X_d \subseteq \mathcal{D}$, $X_t \subseteq \mathcal{T}$
n_D, n_T	Numbers of unique drugs and targets
n	Number of observed data, usually $n >> n_D + n_T$
$x = (d, t)$	Input variable, drug-target pair
$y \in \mathbb{R}^n$	Labels, label y is assosiated with pair x
$X = (x_1, \ldots, x_n) \in (\mathcal{X}^n)^\top$	Set of inputs
$Y = (y_1, \ldots, y_n)^\top \in \mathbb{R}^n$	Set of labels
$\mathcal{L}$	Loss function
$f : \mathcal{X} \to \mathbb{R}$	Prediction function
$p = f(X) \in \mathbb{R}^n$	Predictions
$a \in \mathbb{R}^n$	Dual coefficient (variable)
$J(.)$	Objective function for optimization
$J_1(a)$, $J_2(a)$	DC components of J, i.e. $J(a) = J_1(a) - J_2(a)$
$\nabla J(a)$	Gradient of J at a
$\partial J(a)$, $\partial_c J(a)$	Subdifferentials of J at a
ξ	(Sub)gradient, $\xi \in \partial J(a)$
$\mathcal{C}$	Regularization function
λ, ρ, ρ_1, ρ_2	Regularization parameters
k	(Maximum) number of non-zero elements, $k \in \{1, \ldots, n\}$
$\|a\|_2$, $\|a\|_1$, $\|a\|_0$	ℓ_2-norm (Euclidean), ℓ_1-norm, and ℓ_0-pseudo-norm
$\|a\|_{[k]}$	Polyhedral k-norm
$k_{\mathcal{D}} : \mathcal{D} \times \mathcal{D} \to \mathbb{R}$, $k_{\mathcal{T}} : \mathcal{T} \times \mathcal{T} \to \mathbb{R}$	Kernel functions for drugs and targets
$k_{\mathcal{D},\mathcal{T}} : (\mathcal{D} \times \mathcal{T}) \times (\mathcal{D} \times \mathcal{T}) \to \mathbb{R}$	Pairwise kernel function (also Kronecker product kernel)
$K \in \mathbb{R}^{n \times n}$	Pairwise kernel matrix for training, $K_{i,j} = k_{\mathcal{D},\mathcal{T}}((d_i, t_i), (d_j, t_j))$, $i, j = 1, \ldots, n$
S1–S4	Experimental settings

where "conv" is the convex hull of a set and each vector $\xi \in \partial J(a)$ is called a *subgradient*. By using this subdifferential, it is easy to derive a necessary optimality condition for $a^* \in \mathbb{R}^n$ to be a local optimizer, namely, $0 \in \partial J(a^*)$ [3]. Any point satisfying this condition is called *stationary*. Note that stationarity can also be a sufficient condition for global optimality, but only when the considered function is convex.

2.2 DC programming

In DC programming, we consider problems where all functions can be stated as a difference of two convex (DC) functions. The general unconstrained DC problem is given with

$$\underset{a \in \mathbb{R}^n}{\text{minimize}} \quad J(a), \tag{1}$$

where $J : \mathbb{R}^n \to \mathbb{R}$ is DC. Formally, a function $J : \mathbb{R}^n \to \mathbb{R}$ is *DC* if there exists two convex functions $J_1, J_2 : \mathbb{R}^n \to \mathbb{R}$ enabling us to write

$$J(a) = J_1(a) - J_2(a) \quad \text{for all } a \in \mathbb{R}^n.$$

In what follows, $J_1 - J_2$ is called a *DC decomposition*, where the separate convex functions J_1 and J_2 are *DC components*. Note that one or both of the DC components can be nonsmooth and that a DC function has infinitely many different DC decompositions. Altogether DC functions constitute an important and wide subclass of nonsmooth nonconvex functions including, for example, every convex, concave, and twice continuously differentiable function [19, 21].

Before stating some optimality conditions for the DC problem (1), we need to introduce a subdifferential for convex DC components $J_i : \mathbb{R}^n \to \mathbb{R}$, $i = 1, 2$. The *subdifferential* at a point $a \in \mathbb{R}^n$ is given as a set [2, 3]

$$\partial_c J_i(a) = \left\{ \xi \in \mathbb{R}^n : \ J_i(b) - J_i(a) \geq \xi^\top (b - a) \text{ for all } b \in \mathbb{R}^n \right\}, \quad i = 1, 2.$$

Each vector $\xi^i \in \partial_c J_i(a)$ is called a *subgradient* of J_i at a for $i = 1, 2$. From the above definitions, we see that $\partial_c J_i(a) = \partial J_i(a)$ for $i = 1, 2$. Thus, to simplify the notations we denote the subdiffereantial of the convex DC component J_i, $i = 1, 2$, also with the Clarke subdifferential $\partial J_i(a)$.

For the problem (1), the most commonly used necessary optimality condition is *criticality* requiring that at a point $a^* \in \mathbb{R}^n$ the condition

$$\partial J_1(a^*) \cap \partial J_2(a^*) \neq \emptyset$$

is fulfilled. Note that a stationary point is always critical. However, the inverse result does not necessarily hold. This is due to the fact that at a point $a \in \mathbb{R}^n$ for a DC function $J = J_1 - J_2$ we can only guarantee a result [3]

$$\partial J(a) \subseteq \partial J_1(a) - \partial J_2(a) \tag{2}$$

and equality in the relationship happens only when J_1 or J_2 is differentiable. Thus, only differentiability of J_1 or J_2 together with criticality ensures also stationarity. In DC programming algorithms, the goal is typically find only a critical point [23, 27, 28].

2.3 ℓ_0-pseudo-norm and sparse optimization

Optimization problems seeking sparsity of the solutions have recently received broad attention. They are applied, for instance, in regression analysis (see, e.g. [4, 11, 34]), sparse principal component analysis (see, e.g. [17, 53, 55]), and signal and image processing (see, e.g. [24, 30, 54]). In this subsection, we recall the DC formulation for sparse optimization problem [11, 13].

In sparse optimization, we consider a cardinality-constrained optimization problem:

$$\begin{cases} \underset{a \in \mathbb{R}^n}{\text{minimize}} & J(a) \\ \text{subject to} & \|a\|_0 \leq k, \end{cases} \tag{3}$$

where $J : \mathbb{R}^n \to \mathbb{R}$, $\|a\|_0$ is the ℓ_0-*pseudo-norm* that simply counts the number of nonzero components of a vector, and $k \in \{1, \ldots, n\}$ represents the maximum allowed number of non-zero elements. Due to the non-convex and discontinuous nature of the ℓ_0-pseudo-norm, solving the problem (3) is known to be NP-hard [35]. As a result, the ℓ_0-pseudo-norm in (3)

is often replaced with its tight convex relaxation, the ℓ_1-norm [11]. However, this kind of relaxation deteriorates the control over the desired sparsity level and, thus, we propose taking advantage of the continuous formulation of the ℓ_0-pseudo-norm [11, 13]. We rely to the class of *polyhedral k-norms* $\|a\|_{[k]}$ defined as the sum of k largest components (in modulus) of the vector a. The fact that polyhedral k-norms are intermediate between ℓ_1- and ℓ_∞-norms, allows us to derive the following equivalence, valid for $1 \leq k \leq n$:

$$\|a\|_0 \leq k \iff \|a\|_1 - \|a\|_{[k]} = 0.$$

It is noteworthy that the cardinality-constraint in (3) is defined by a discontinuous function whereas the exact reformulation $\|a\|_1 - \|a\|_{[k]}$ is a continuous one, although both describe the same feasible set [11, 13].

We remind that $\|a\|_1 - \|a\|_{[k]} \geq 0$ for all $a \in \mathbb{R}^n$ and, thus, we transfer the cardinality-constrained problem (3) into a penalty function formulation:

$$\underset{a \in \mathbb{R}^n}{\text{minimize}} \quad J(a) + \rho \left(\|a\|_1 - \|a\|_{[k]} \right) \tag{4}$$

with a penalty parameter $\rho > 0$. Assuming that the original objective function J in (4) is a DC function with the decomposition $J = J_1 - J_2$ (note that a convex function is trivially DC with $J_1 = J$), formula (4) can be written as a DC optimization problem:

$$\underset{a \in \mathbb{R}^n}{\text{minimize}} \quad \hat{J}(a) = \hat{J}_1(a) - \hat{J}_2(a),$$

where $\hat{J}_1(a) = J_1(a) + \rho\|a\|_1$ and $\hat{J}_2(a) = J_2(a) + \rho\|a\|_{[k]}$.

3 Limited memory bundle DC algorithm

In this section, we propose a novel LMB- DCA for solving the large nonsmooth DC optimization problem (1). The backbone of the new method is the LMBM [14, 16] developed for general large-scale nonsmooth optimization. The LMBM is a bundle-type method characterized by the use of serious and null steps together with the simple aggregation of subgradients. In addition, the search direction is calculated by the limited memory variable metric approach [5] (the limited memory BFGS (L-BFGS) update after a serious step and the limited memory SR1 (L-SR1) update, otherwise). Thus, the LMBM avoids solving the time-consuming quadratic direction finding problem appearing in standard bundle methods (see, e.g. [2, 26]) as well as storing and manipulating large matrices as is the case in the standard variable metric methods. These aspects make the LMBM suitable for solving large-scale nonsmooth optimization problems. Namely, the number of operations needed for the calculation of the search direction and the aggregate values is only linearly dependent on the number of variables.

The basic idea of the new LMB- DCA is very simple: we use the original LMBM, but instead of a subgradient $\xi \in \partial J(a)$ at a point $a \in \mathbb{R}^n$ we substitute it with the subgradient difference $\xi_1 - \xi_2$, where $\xi_1 \in \partial J_1(a)$ and $\xi_2 \in \partial J_2(a)$, unless we encounter an issue. The issue may arise when the difference of subgradients of the DC components does not belong to the subdifferential of the original function (see, Eq. (2)). In practice, this is noted by the failure in the line search procedure or by many consecutive null steps. In these cases, we switch to solve a subproblem similar to the one used in DCA [27, 41]. In what follows, we describe different components of the LMB- DCA in more detail.

The fundamental idea behind the LMBM is to calculate the search direction using the classical limited memory variable metric scheme $d^h = -G^h \nabla J(a^h)$, where G^h is the limited

memory variable metric update that represents the approximation of the inverse of the Hessian matrix, $\nabla J(a^h)$ is a gradient of function J at a point a^h, and h is an iteration index. However, since for a general nonsmooth objective J the gradient does not necessarily exist, an aggregate subgradient $\tilde{\xi}^h \in \partial J(a^h)$ is employed instead in the LMBM. Here, since also the subgradient of the DC function may be unknown, we go one step further and employ the (aggregate) subgradient difference. Therefore, the *search direction* is given by

$$d^h = -G^h \tilde{\xi}^h, \tag{5}$$

where $\tilde{\xi}^h = \xi_1^h - \xi_2^h$ with $\xi_1^h \in \partial J_1(a^h)$ and $\xi_2^h \in \partial J_2(a^h)$ if the previous step was a serious step. However, if the previous step was a null step, $\tilde{\xi}^h$ is an aggregate subgradient difference (see, Eq. (10)). In what follows, we simply denote the subgradient difference by ξ^h for all $h = 1, 2, \ldots$.

The role of the matrix G^h is to accumulate information from the previous iterations. Note, however, that the large matrix G^h is not formed explicitly. Instead, we store only a few (say $\hat{m}_c$) *correction vectors* obtained at the previous iterations of the algorithm and use these vectors to implicitly define the product $G^h \tilde{\xi}^h$. In the smooth case, the correction vectors are given by $s^h = a^{h+1} - a^h$ and $u^h = \nabla J(a^{h+1}) - \nabla J(a^h)$. Nevertheless, in the LMBM — as well as in the proposed LMB- DCA — we may have $a^{h+1} = a^h$ due to the usage of null steps and we use an auxiliary point b^{h+1} (described in detail below) instead of a^{h+1} when updating s^h. In addition, since the (sub)gradient does not necessarily exist for nonsmooth DC objective the correction vectors u^h are computed using the difference of the subgradients of DC components, that is, the vectors are given by $s^h = b^{h+1} - a^h$ and $u^h = \xi^{h+1} - \xi^m$, where m is the index of the last serious step. These correction vectors are append into correction matrices S^h and U^h, respectively. If $h > \hat{m}_c$, we first delete the oldest correction vectors from these matrices.

The search direction d^h in (5) is not necessarily a descent one.[1] Thus, we apply the *line search procedure* that generates a null step instead of a serious step whenever necessary or indicates that neither the condition for a serious step nor the condition for a null step can be reached. We first determine a new auxiliary point $b^{h+1} = a^h + t_R^h d^h$, where $t_R^h \in (0, t_{max}]$ is a step size with an upper bound $t_{max} > 1$. A necessary condition for a *serious step* is to have

$$J(b^{h+1}) \leq J(a^h) - \varepsilon_L t_R^h w^h, \tag{6}$$

where $\varepsilon_L \in (0, 1/2)$ is a line search parameter, and $w^h > 0$ represents the desirable amount of descent of J at a^h. If the condition (6) is satisfied, we set $a^{h+1} = b^{h+1}$ and call this step a serious step. If the current search direction is not good enough, that is, the value of the objective at the auxiliary point b^{h+1} is not decreased enough, either the *null step* occurs or we need to enter the DCA procedure since there is an issue during the line search procedure. In the case of a null step, we have

$$-\beta^{h+1} + (\xi^{h+1})^\top d^h \geq -\varepsilon_R w^h, \tag{7}$$

where $\varepsilon_R \in (\varepsilon_L, 1/2)$ is a line search parameter, ξ^{h+1} is the subgradient difference computed at b^{h+1}, and β^{h+1} is the subgradient locality measure given by

$$\beta^{h+1} = \max\left\{ \left| J(a^h) - J(b^{h+1}) + (s^h)^\top \xi^{h+1} \right|, \gamma \|s^h\|^2 \right\} \tag{8}$$

with the distance measure parameter $\gamma > 0$. In addition, we set $a^{h+1} = a^h$ but information about the objective function is increased since we store the auxiliary point b^{h+1} and the

[1] This is a common feature in nonsmooth optimization.

corresponding auxiliary subgradient difference ξ^{h+1}, and use them to compute the new aggregate values. More precisely, if the condition (7) is satisfied and the maximum number of consecutive null steps is not yet reached (implying that we do not have serious problems with the subgradient differences), we proceed to the aggregation procedure similar to that of the original LMBM — and very different from the one used with standard bundle methods (see, e.g. [2, 26]). That is, we determine multipliers λ_i^h ($i \in \{1, 2, 3\}$) minimizing the function

$$\begin{aligned}\varphi(\lambda_1, \lambda_2, \lambda_3) = [\lambda_1\xi^m + \lambda_2\xi^{h+1} + \lambda_3\tilde{\xi}^h]^\top G^h[\lambda_1\xi^m + \lambda_2\xi^{h+1} + \lambda_3\tilde{\xi}^h] \\ + 2(\lambda_2\beta^{h+1} + \lambda_3\tilde{\beta}^h),\end{aligned} \tag{9}$$

and satisfying $\lambda_i^h \geq 0$ and $\sum_{i=1}^3 \lambda_i^k = 1$. Here, ξ^m is the subgradient difference at the current point $a^m = a^h$ (m denotes the iteration index of the last serious step), ξ^{h+1} is the subgradient difference at the auxiliary point b^{h+1}, and $\tilde{\xi}^h$ is the current aggregate subgradient difference from the previous iteration ($\tilde{\xi}^1 = \xi_1^1 - \xi_2^1$). In addition, β^{h+1} is the current subgradient locality measure and $\tilde{\beta}^h$ is the current aggregate subgradient locality measure ($\tilde{\beta}^1 = 0$). The new aggregate values can then be computed as

$$\tilde{\xi}^{h+1} = \lambda_1^h\xi^m + \lambda_2^h\xi^{h+1} + \lambda_3^h\tilde{\xi}^k \quad \text{and} \quad \tilde{\beta}^{h+1} = \lambda_2^h\beta^{h+1} + \lambda_3^h\tilde{\beta}^h. \tag{10}$$

This simple aggregation procedure gives us the possibility to retain the global convergence without solving the complicated quadratic direction finding problem appearing in standard bundle methods (see, e.g. [2]). Moreover, only one trial point b^{h+1} and the corresponding subgradient difference ξ^{h+1} need to be stored instead of $n + 3$ subgradients typically stored in bundle methods. Finally, we point out that the aggregate values need to be computed only if the last step was a null step. Otherwise, we set $\tilde{\xi}^{h+1} = \xi^{h+1}$ and $\tilde{\beta}^{h+1} = 0$.

On the other hand, if the condition (7) is not satisfied during the line search procedure or the maximum number of consecutive null steps is met, we solve the convex subproblem similar to the DCA to escape the current problematic iteration point. This yields a problem

$$\underset{a\in\mathbb{R}^n}{\text{minimize}} \quad \bar{J}_h(a) = J_1(a) - [J_2(a^h) + (\xi_2^h)^\top(a - a^h)], \tag{11}$$

where $\xi_2^h \in \partial J_2(a^h)$, which is also solved with LMBM. This means that in the subproblem we are linearizing the second DC component J_2 at the current iteration point a^h. One main benefit of this is that the solution a^{h+1} of the subproblem (11) always satisfies

$$J(a^{h+1}) \leq \bar{J}_h(a^{h+1}) \leq \bar{J}_h(a^h) = J(a^h).$$

Thus, we should always use a^h as a starting point in the LMBM when we solve (11). Furthermore, if $\bar{J}_h(a^{h+1}) < \bar{J}_h(a^h)$ we know that a better solution is found for the original objective J and a serious step can be performed. However, if $\bar{J}_h(a^{h+1}) = \bar{J}_h(a^h)$ is satisfied we know that $0 \in \partial\bar{J}_h(a^h)$ implying that $\xi_2^h \in \partial J_1(a^h)$. This guarantees the criticality of the current iteration point a^h and we can terminate the execution of the whole LMB- DCA with a^h as its solution.

Remark 1 The original LMBM is globally convergent for locally Lipschitz continuous functions [16]. Therefore, if $\xi_1^h - \xi_2^h \in \partial J(a^h)$ for all h also the proposed algorithm converges to a Clarke stationary point. This is the case, for instance, if either J_1 or J_2 is a smooth function.

Remark 2 If we stop at Step 3 of the LMB-DCA with $m = h$ (i.e. the previous step was a serious step), we have found a point a^h that (approximately) satisfies $0 = \xi^h = \xi_1^h - \xi_2^h$. Thus, we can say a^h is a critical point.

Algorithm 1 LMB- DCA

Input: A starting point $a^1 \in \mathbb{R}^n$, the final accuracy tolerance $\varepsilon > 0$, descent parameters $\varepsilon_L \in (0, 1/2)$ and $\varepsilon_R \in (\varepsilon_L, 1/2)$, the upper bound $t_{\max} > 1$ for serious steps, the distance measure parameter $\gamma > 0$, the number of stored corrections $\hat{m}_c \geq 3$, and the maximum number of consecutive null steps $n^0_{\max} \geq 0$.

Step 0. (*Initialization*) Compute $J(a^1) = J_1(a^1) - J_2(a^1)$ and $\xi^1 = \xi^1_1 - \xi^1_2$, where $\xi^1_1 \in \partial J_1(a^1)$ and $\xi^1_2 \in \partial J_2(a^1)$. Initialize the correction matrices S^1 and U^1 as empty matrices and set $b^1 = a^1$, $\beta^1 = 0$, and $h = 1$.

Step 1. (*Serious step initialization.*) Set $\tilde{\xi}^h = \xi^h$ and $\tilde{\beta}^h = 0$. Set $m = h$.

Step 2. (*Direction finding*) If $h = 1$, set $d^1 = -\xi^1$ and go to Step 3. Else, compute

$$d^h = -G^h \tilde{\xi}^h$$

by the L-BFGS update if $m = h$ (use at most $\hat{m}_c$ correction vectors in S^h and U^h) and by the L-SR1 update, otherwise.

Step 3. (*Stopping criterion*) Compute

$$w^h = -(\tilde{\xi}^h)^\top d^h + 2\tilde{\beta}^h.$$

If $w^h < \varepsilon$, then **stop** with a^h as the final solution.

Step 4. (*Line search and auxiliary step*) Determine the step size $t^h_R \in (0, t_{\max}]$. Evaluate

$$b^{h+1} = a^h + t^h_R d^h \quad \text{and} \quad \xi^{h+1} = \xi^{h+1}_1 - \xi^{h+1}_2,$$

where $\xi^{h+1}_1 \in \partial J_1(b^{h+1})$ and $\xi^{h+1}_2 \in \partial J_2(b^{h+1})$. Set $s^h = b^{h+1} - a^h = t^h_R d^h$ and $u^h = \xi^{h+1} - \xi^m$ (recall that m is the index of the last serious step). Append these values to S^h and U^h and denote the results as S^{h+1} and U^{h+1}, respectively.

Step 5. (*Serious step*) If

$$J(b^{h+1}) - J(a^h) \leq -\varepsilon_L t^h_R w^h,$$

set $a^{h+1} = b^{h+1}$, $J(a^{h+1}) = J(b^{h+1})$, $\beta^{h+1} = 0$, $h = h + 1$, and go to Step 1.

Step 6. (*Null step*)
Calculate the locality measure β^{h+1} by (8). If $h - m < n^0_{\max}$ and $-\beta^{h+1} + (d^h)^\top \xi^{h+1} \geq -\varepsilon_R w^h$, set $a^{h+1} = a^h$ and go to Step 7. Else, go to Step 8.

Step 7. (*Aggregation.*) Determine multipliers $\lambda^h_i \geq 0$ for all $i \in \{1, 2, 3\}$, $\sum_{i=1}^3 \lambda^k_i = 1$ that minimize the function (9), where G^h is calculated by the same updating formula as in Step 2.
Compute $\tilde{\xi}^{h+1}$ and $\tilde{\beta}^{h+1}$ as in (10). Set $h = h + 1$ and go to Step 2.

Step 8. (*DCA step*) Find a solution a^{h+1} to the convex optimization problem (11). If $J(a^{h+1}) = J(a^h)$, then **stop** with a^h (or a^{h+1}) as the final solution. Otherwise compute $\xi^{h+1} = \xi^{h+1}_1 - \xi^{h+1}_2$ with $\xi^{h+1}_1 \in \partial J_1(a^{h+1})$ and $\xi^{h+1}_2 \in \partial J_2(a^{h+1})$. After this set $s^h = a^{h+1} - a^h$ and $u^h = \xi^{h+1} - \xi^m$ and append these values to S^h and U^h and denote the results as S^{h+1} and U^{h+1}, respectively (i.e. overwrite the previous update made in Step 4). Update $h = h + 1$ and go to Step 1.

Remark 3 The DCA is proved to be globally convergent to a critical point [27, 41]. In principle, the DCA just successively solves the subproblem (11) and updates the iteration point. Thus, if we enter Step 8 of the LMB-DCA infinitely many times, the proposed algorithm converges to a critical point due to the repeated execution of the DCA step. In addition, if the execution of the LMB-DCA is stopped at Step 8, then we directly have a critical point.

Remark 4 To guarantee the global convergence in the original LMBM, the sequence $\{w^h\}$ has to be nonincreasing in consecutive null steps and matrices $(D^i)^{-1}$ for all $i = 1, \ldots, h$ need to be bounded. This leads to a more complex algorithm that is described in detail in [16].

4 Preliminaries for pairwise kernel learning

In this section, we provide the theoretical background on pairwise learning problems and kernel methods.

4.1 Pairwise learning problem

The goal of supervised learning (e.g., regression and classification) is to learn an unknown prediction function $f : \mathcal{X} \to \mathcal{Y}$ from a set of training samples $\{(x_i, y_i)\}_{i=1}^n$ each consisting of an input $x_i \in \mathcal{X}$ and its corresponding label $y_i \in \mathcal{Y}$ such that f can correctly predict a label for a new object $x \in \mathcal{X}$ unseen during the training phase. Here, the label space is $\mathcal{Y} = \mathbb{R}$ for regression and $\mathcal{Y} = \{0, 1\}$ for classification problems. In what follows, we consider almost solely the regression problems.

In pairwise learning, the training input consists of pairs of objects $x = (d, t)$ and their real valued labels y. We call these objects drugs $d \in \mathcal{D}$ and targets $t \in \mathcal{T}$, but they can represent various other objects in different applications. Different fields often consider divergent but related pairwise prediction tasks.[2] Nevertheless, the two most fundamental assumptions of pairwise learning are given below [48].

Assumption 1 *(Pairwise data assumption)* Both the drugs and the targets tend to appear several times as parts of different inputs in an observed data set.

Assumption 2 *(Nonlinearity assumption)* The predicting functions are usually not linear combinations of functions depending only on drugs or targets.

Assumption 1 means that the same drug d_i may belong to two (or more) different samples (d_i, t_j) and (d_i, t_k) while Assumption 2 means that we do not have global ordering of drugs (or targets). More precisely, the opposite of Assumption 2 would mean that if $f(d, t) = f_d(d) + f_t(t)$ for some functions f_d and f_t, then $f(d_i, t_j) > f(d_k, t_j)$ would imply $f(d_i, t_l) > f(d_k, t_l)$ for all drugs and targets. In other words, there would always be a single drug that would be the best choice for all targets (and vice versa).

We formulate the problem of learning the prediction function f as the regularized empirical risk minimization problem

$$f^* = \arg\min_{f \in \mathcal{H}} \mathcal{L}(p, y) + \lambda \mathcal{C}(f), \tag{12}$$

where $p = f(X) \in \mathbb{R}^n$ are the predicted and $y \in \mathbb{R}^n$ are the correct outputs for the training set, $X = (x_1, \ldots, x_n) \in (\mathcal{X}^n)^\top$ is the set of inputs, $\mathcal{L}$ is a convex non-negative loss function, and $\lambda > 0$ and $\mathcal{C}$ are the regularization parameter and function, respectively. In addition, $\mathcal{H}$ is the hypothesis space.

The standard formulation of problem (12) is called RLS or ridge regression. It is obtained by choosing a squared loss to loss function $\mathcal{L}(p, y) = \|y - p\|^2$, a Euclidean norm for

[2] This leads to the need for different experimental settings, see [39, 48] and Sect. 7.1.

regularization $\lambda\mathcal{C}(f) = \frac{\lambda}{2}\|f\|^2_{\mathcal{H}}$, and the reproducing kernel Hilbert space (RKHS) as the hypothesis space $\mathcal{H}$.

4.2 Kernel methods

Nowadays kernel methods are the de facto methods in supervised learning [1]. They provide a powerful framework for learning complex patterns in data by implicitly mapping it into a higher-dimensional feature space [22]. Kernel methods can be used when the training data either have explicit feature vectors or implicit feature vectors are defined via positive semi-definite kernel functions. Furthermore, when both drugs and targets have their own feature representation or kernel, we can use a pairwise kernel to define a kernel for the pair. The simplest pairwise kernel is obtained by concatenating the feature vectors of drugs and targets and applying a standard kernel (like polynomial kernel) into this combined feature vector. However, the simplest kernel that models actual pairwise interactions between drug and target features is the pairwise Kronecker product kernel [48]. It also allows generalization for such new pairs where neither the drug nor the target occur in the training set — a setting known as zero-shot learning [1].

Let $k_{\mathcal{D}} : \mathcal{D} \times \mathcal{D} \to \mathbb{R}$ and $k_{\mathcal{T}} : \mathcal{T} \times \mathcal{T} \to \mathbb{R}$ denote positive semidefinite kernel functions[3] defined for drugs and targets. Then the Kronecker product kernel for the drug-target pairs $k_{\mathcal{D},\mathcal{T}} : (\mathcal{D} \times \mathcal{T}) \times (\mathcal{D} \times \mathcal{T}) \to \mathbb{R}$ is defined as a product of these two base kernels. That is,

$$k_{\mathcal{D},\mathcal{T}}((d,t),(d',t')) = k_{\mathcal{D}}(d,d')k_{\mathcal{T}}(t,t').$$

Further, let $K \in \mathbb{R}^{n\times n}$ denote the pairwise Kroneker product kernel matrix containing all the kernel evaluations between the drug-target pairs such that $K_{i,j} = k_{\mathcal{D},\mathcal{T}}((d_i,t_i),(d_j,t_j))$, $i,j = 1,\ldots,n$. By choosing RKHS associated with $k_{\mathcal{D},\mathcal{T}}$ as the hypothesis space, the generalized representer theorem [44] implies that the minimizing function f in (12) can be given as:

$$f(d,t) = \sum_{i=1}^{n} a_i k_{\mathcal{D},\mathcal{T}}((d_i,t_i),(d,t)),$$

where a_i is a component of the vector $a \in \mathbb{R}^n$ of dual coefficients and index i goes trough all observed drug-target pairs. In addition, the predictions for the training data can be given with the kernel matrix as $p = Ka$ [48].

It is worth noting that Assumption 1 implies that if n is the number of observed data and n_D and n_T are the numbers of unique drugs and targets in the data, respectively, then $n >> n_D + n_T$. This fact is used to develop computational short-cuts tailored specifically to pairwise learning problems in [1, 48]. More precisely, the large kernel matrix K need not be computed explicitly but the matrix–vector product Ka can be computed implicitly in $\mathcal{O}(n\,(n_D + n_T))$ time using a sparse Kronecker product multiplication algorithm known as the *generalized vec trick* (GVT). This computational shortcut can be applied to speed up any optimization approach whose computational complexity is dominated by multiplications of a pairwise kernel matrix with a vector.

[3] In this work, the kernel functions $k_{\mathcal{D}}$ for drugs and $k_{\mathcal{T}}$ for targets come from the chemical structure and sequence similarity matrices, respectively.

5 Model formulation

In this section, we introduce the model we are using. We consider the regularized empirical risk minimization problem (12) with various loss functions described in detail in Sect. 5.2. In addition, instead of just replacing the most commonly used squared regularization term with the ℓ_0-pseudo-norm as in [37], we apply a double regularization scheme. It is worth mentioning that we end up with a nonsmooth DC optimization problem.

5.1 Double regularization

As pointed out in Sect. 4, the standard formulation of the pairwise kernel learning problem (12) is to use the squared loss with the squared regularization. In [37], the squared regularization is replaced with the ℓ_0-pseudo-norm. However, the direct use of the ℓ_0-pseudo-norm may lead to a situation where the largest nonzero dual coefficients are left without any regularization. That is, in the continuous representation of the ℓ_0-pseudo-norm the polyhedral k-norm cancels out the k largest dual coefficients from the ℓ_1-norm. Therefore, we use the ℓ_1-norm as a double regularization to provide regularization for all dual coefficients. The ℓ_1-norm is selected (instead of the usually used ℓ_2-norm) as it supports our effort to produce sparse solutions.

We consider the double regularized empirical risk minimization problem in a form

$$\underset{a\in\mathbb{R}^n}{\text{minimize}} \quad J(a) = \mathcal{L}(Ka, y) + \rho_1\|a\|_1 + \rho_2(\|a\|_1 - \|a\|_{[k]}) \tag{13}$$

with $\rho_1, \rho_2 > 0$. This formula is both nonsmooth and nonconvex. However, function $J(a)$ can be written as a DC function, which allows us to compute the subgradients of convex DC components separately. The DC formulation of the problem (13) is

$$\underset{a\in\mathbb{R}^n}{\text{minimize}} \quad J(a) = J_1(a) - J_2(a),$$

where $J_1(a) = \mathcal{L}(Ka, y) + (\rho_1 + \rho_2)\|a\|_1$ and $J_2(a) = \rho_2\|a\|_{[k]}$.

The (initial) double regularization parameters ρ_1 and ρ_2 in (13) are given by the formula

$$\rho_1 = \rho_2 = \frac{1}{n^2} \cdot \frac{\mathcal{L}(p^1, y)}{\|a^1\|_1},$$

where $a^1 \in \mathbb{R}^n$ is the user-specified starting point and $\mathcal{L}(p^1, y)$ is the value of the loss function in that point (remind that $p^1 = Ka^1$). Parameter ρ_1 is fixed while parameter ρ_2 is used as a penalty parameter and increased at every iteration of the proposed SparsePKL algorithm until the feasible solution with respect to the desired sparsity level is obtained. More details of this will be given in Sect. 6.

5.2 Loss functions and their (sub)gradients

We use the following loss functions with (13):

- Squared loss: $\mathcal{L}(p, y) = \frac{1}{2}\sum_{i=1}^n (p_i - y_i)^2$;
- Squared epsilon-intensive loss: $\mathcal{L}(p, y) = \frac{1}{2n}\sum_{i=1}^n \max(0, |p_i - y_i| - \varepsilon)^2$, where $\varepsilon > 0$;
- Epsilon-intensive squared loss: $\mathcal{L}(p, y) = \frac{1}{2n}\sum_{i=1}^n \max(0, (p_i - y_i)^2 - \varepsilon)$, where $\varepsilon > 0$;
- Epsilon-intensive absolute loss: $\mathcal{L}(p, y) = \frac{1}{n}\sum_{i=1}^n \max(0, |p_i - y_i| - \varepsilon)$, where $\varepsilon > 0$;
- Absolute loss: $\mathcal{L}(p, y) = \frac{1}{n}\sum_{i=1}^n |p_i - y_i|$.

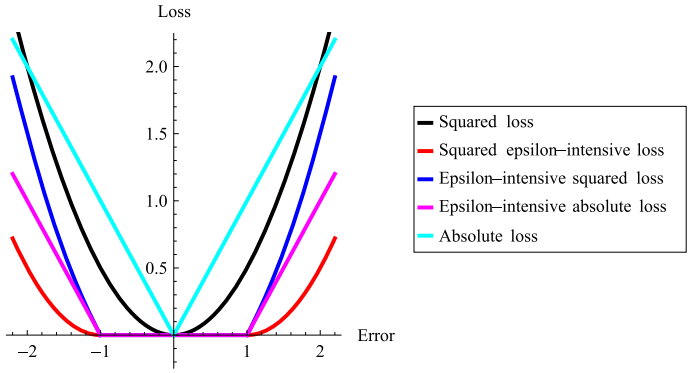


Fig. 1 Illustration of different loss functions with the selection $\varepsilon = 1$

These loss functions are illustrated in Fig. 1.

The gradients $\zeta = \nabla\mathcal{L}(p, y)$ and subgradients $\zeta \in \partial\mathcal{L}(p, y)$ of the above-mentioned loss functions with respect to the predictions are given below. Note that it is enough to find one subgradient ζ from the subdifferential $\partial\mathcal{L}(p, y)$ at each point p to apply a nonsmooth optimization method.

- Squared loss: $\zeta_i = p_i - y_i, i = 1, \ldots, n$.
- Squared epsilon-intensive loss: $\zeta_i = \begin{cases} 0, & \text{if } |p_i - y_i| < \varepsilon, \\ p_i - y_i - \varepsilon)/n, & \text{if } p_i - y_i \geq \varepsilon, \\ (p_i - y_i + \varepsilon)/n, & \text{otherwise.} \end{cases}$
- Epsilon-intensive squared loss: $\zeta_i = \begin{cases} 0, & \text{if } (p_i - y_i)^2 < \varepsilon, \\ (p_i - y_i)/n, & \text{otherwise.} \end{cases}$
- Epsilon-intensive absolute loss: $\zeta_i = \begin{cases} 0, & \text{if } |p_i - y_i| < \varepsilon, \\ 1/n, & \text{if } p_i - y_i \geq \varepsilon, \\ -1/n, & \text{otherwise.} \end{cases}$
- Absolute loss: $\zeta_i = \begin{cases} 1/n, & \text{if } p_i - y_i > 0, \\ 0, & \text{if } p_i - y_i = 0, \\ -1/n, & \text{otherwise.} \end{cases}$

As pointed out the gradients and subgradients above are given with respect to predictions p. In the case of Kronecker product kernel methods, the loss function is $\mathcal{L}(p, y)$, where $p = Ka$. Due to this, the loss function can be written in the form $f = g \circ h$ where $h : \mathbb{R}^n \to \mathbb{R}^n$ and $g : \mathbb{R}^n \to \mathbb{R}$ are given with formulas $h(a) = Ka$ and $g(p) = \mathcal{L}(p, y)$. In addition, h is linear and continuously differentiable, whereas g is convex and thus subdifferentially regular. These properties guarantee that the (sub)differential of f with respect to a can be computed via (generalized, see e.g. [3] Theorem 3.20) chain rule as

$$\partial f(a) = \nabla h(a)^\top \partial g(h(a)) = K^\top \partial\mathcal{L}(p, y).$$

This means that $K^\top\zeta$, where $\zeta = \nabla\mathcal{L}(p, y)$ or $\zeta \in \partial\mathcal{L}(p, y)$, is a (sub)gradient of the loss function with respect to the dual variable a.

5.3 Subgradient of double regularization

The final point in this section is to give subgradients of the regularization part in (13). A subgradient η of $(\rho_1 + \rho_2)\|a\|_1$ with respect to the dual variable a is given by

$$\eta_i = \begin{cases} (\rho_1 + \rho_2), & \text{if } a_i > 0, \\ -(\rho_1 + \rho_2), & \text{if } a_i < 0, \\ 0, & \text{otherwise.} \end{cases}$$

Next, let us denote by $\mathcal{I}_{[k]} = \{i_1, \ldots, i_k\} \subseteq \{1, 2, \ldots, n\}$ the index set of k largest (in modulus) components of a. The subgradient ϑ of $\rho_2\|a\|_{[k]}$ with respect to the dual variable a is given by

$$\vartheta_i = \begin{cases} \rho_2, & \text{if } a_i \geq 0 \text{ and } i \in \mathcal{I}_{[k]}, \\ -\rho_2, & \text{if } a_i < 0 \text{ and } i \in \mathcal{I}_{[k]}, \\ 0, & \text{otherwise.} \end{cases}$$

When we combine the previous information, the subgradients $\xi_1 \in J_1(a)$ and $\xi_2 \in J_2(a)$ are obtained in the model (13) with respect to the dual variable a using the formulas

$$\xi_1 = K^\top \zeta + \eta \quad \text{and} \quad \xi_2 = \vartheta.$$

Furthermore, utilizing these subgradients we obtain the subgradient difference $\xi_1 - \xi_2$ for the objective J in (13). This is enough for calculations since the proposed method LMB-DCA does not need the exact subgradient of J.

6 Sparse pairwise kernel learning

In this section, we propose the pairwise kernel learning algorithm SPARSEPKL as Algorithm 2 and give some details of its implementation.

6.1 Pairwise kernel learning algorithm

The pairwise kernel learning algorithm SPARSEPKL aims to produce predictions with the desired sparsity of the solution. This is done by solving the double regularized risk minimization problem (13) with an increasing penalty parameter ρ_2. With ρ_2 large enough, the problem (13) has a solution a_{best} which is feasible with respect to the ℓ_0-pseudo-norm constraint $\|a\|_0 \leq k$ (see, Eq. (3)). Nevertheless, if the desired sparsity level is not achieved within the maximum number of iterations given by the user, the SPARSEPKL algorithm returns the sparsest infeasible solution obtained so far (denoted by a_{sis}) with the warning.

The above-mentioned double regularized risk minimization problem (13) is solved with the LMB- DCA. To accelerate the solution process the problem is solved using a little bit smaller value of k than the user-specified sparsity level would imply (see Steps 0 and 1 of Algorithm 2). The reason for this is that the penalty-based optimization methods tend to slow down considerably near the boundary of the feasible region. Naturally, we accept the solution whenever the desired sparsity level is achieved (see Step 3 of Algorithm 2).

[4] In this work Gaussian kernels for drugs and targets are used but other kernel functions can be applied as well. The pairwise interactions between drug and target features are modelled via the pairwise Kronecker product kernel matrix K.

Algorithm 2 SPARSEPKA

Input: Give data $x = (d, t) \in \mathcal{X}$ with labels $y \in \mathcal{Y}$ and divide it to training, validation, and test sets. Choose the loss function $\mathcal{L}$ and the desired sparsity percentage $nz\%$ describing the amount of nonzero variables. Select the kernel functions $k_{\mathcal{D}}$ and $k_{\mathcal{T}}$ for drugs and targets, respectively[4]. Select $it_{max}^{lmbdca} > 0$, the maximum number of iterations used with LMB- DCA after which the predictions are (re)validated and $it_{max}^{pka} > 0$ the maximum number of iterations used here. Select $\varepsilon > 0$ for epsilon-intensive loss functions (default $\varepsilon = 0.01$) and $\delta^1 > 1$ for penalty parameter updating.

Step 0. (*Initialization*) Set a starting point $a^1 = 1/n \in \mathbb{R}^n$ and the desired number of nonzero components $k = nz\% \cdot n$. Set $k' = 0.99k$. Compute $p^1 = Ka^1$ via GVT and the (initial) double regularization parameters ρ_1 and ρ_2^1 by

$$\rho_1 = \rho_2^1 = \frac{1}{n^2} \cdot \frac{\mathcal{L}(p^1, y)}{\|a^1\|_1}.$$

Set $j = 1$, $CI_{best} = 0$, $nz_{sis} = n$, and the solution feasibility $t = false$.

Step 1. (*Double regularized risk minimization problem*) Apply LMB-DCA (Algorithm 1) to minimize the double regularized risk minimization problem (13) starting from a^j with the regularization parameters $\rho_1 = \rho_1$ and $\rho_2 = \rho_2^j$, and $k = k'$. Stop Algorithm 1 after it_{max}^{lmbdca} iterations unless it has already terminated due to criticality. In the latter case, set $t = true$. Denote the obtained solution by a^{j+1} and the number of nonzero components in it by nz_a.

Step 2. (*Validation*) If $nz_a \leq k$ compute the C-index CI_{val} w.r.t. the validation data and, if $CI_{val} > CI_{best}$, set $CI_{best} = CI_{val}$, $a_{best} = a^{j+1}$, and $nz_{best} = nz_a$. Otherwise, if $nz_a \leq nz_{sis}$, set $nz_{sis} = nz_a$ and $a_{sis} = a^{j+1}$.

Step 3. (*Termination with the feasible solution*) If $j > it_{\max}^{pka}$, set $t = true$. If $t = true$ and $nz_{best} \leq k$, **stop** with a_{best} as a best solution. Compute and return the predictions and C-index w.r.t. test data. Otherwise, set $t = false$.

Step 4. (*Termination with an infeasible solution*) If $j > it_{\max}^{pka}$, **stop** with an infeasible solution a_{sis}. Compute and return the predictions and C-index w.r.t. test data, and the obtained sparsity level nz_{sis} with *warning* of infeasibility.

Step 5. (*Feasibility achieved*) If $nz_a \leq k$, set $\rho_2^{j+1} = \rho_2^j$, $j = j + 1$ and go to Step 1.

Step 6. (*Update penalty parameter*) Set $\rho_2^{j+1} = \delta^j \rho_2^j$, $\delta^{j+1} = \max(1.1, \delta^j/2)$, $j = j + 1$ and go to Step 1.

At every iteration of the SPARSEPKL algorithm the predictions are validated using a separate validation set and an evaluation metric called the *concordance index* (C-index) [12]. The C-index is a rank-based performance measure that is defined as the probability that the predictions for two sample points are in the same order as their real labels. In the case of binary interaction labels, the C-index is equal to the widely used metric area under the receiver operating characteristic curve (AUC) [39].

Remark 5 One might expect that the number of nonzero variables would decrease at each iteration of the SPARSEPKL algorithm. However, this is not necessarily the case in penalty-based methods with changing penalty parameters and, especially, when the DCA step of LMB- DCA is employed (see Step 8 of Algorithm 1) the number of nonzero variables may increase. Therefore, if the desired sparsity is not yet achieved in Step 2 of Algorithm 2, we save the sparsest infeasible solution a_{sis} obtained so far and the number of nonzero components nz_{sis} in it.

6.2 Implementation

The SPARSEPKL algorithm is implemented using a combination of Python and Fortran 95 via the F2PY interface generator. The GVT from RLScore https://github.com/aatapa/RLScore [38] is applied to compute Kronecker product kernels implicitly. The k-norm is computed using the bisection method (mostly) without sorting or copying the data. Here, we employed the source code by PierU available at https://stackoverflow.com/questions/75975549/find-and-modify-highest-n-values-in-a-2d-array-in-fortran. The source code of SPARSEPKL — including LMB- DCA and all the above-mentioned loss functions — is available at https://github.com/napsu/sparsePKL.

7 Numerical experiments

Now we are ready to compare the proposed algorithms SPARSEPKL and LMB- DCA with some state-of-the-art methods in pairwise kernel learning and DC optimization. We start this section by describing the data sets and different experimental settings for data and continue with some implementation details of the algorithms used. Finally, we give the results of our experiments.

7.1 Data and experimental settings

Numerical experiments are run on seven real pairwise drug-target data sets whose characteristics are presented in Table 2. These datasets are selected for three key reasons:

1. Pairwise kernel methods were originally introduced for biomedical interaction prediction and are most widely used in this domain. The datasets used in this study serve as standard benchmarks in the field.
2. The datasets exhibit diverse characteristics, including variations in size, label type (continuous or binary), and sparsity.
3. The datasets provide feature representations for both drugs and targets, which are essential for pairwise learning using the Kronecker kernel. While open-source drug-target interaction (DTI) datasets with these properties are scarce, they are crucial for achieving meaningful generalization in zero-shot learning scenarios.

Each data consists of three matrices $X_d \in \mathbb{R}^{n_D \times n_D}$, $X_t \in \mathbb{R}^{n_T \times n_T}$ and $Y \in \mathbb{R}^{n_D \times n_T}$, where the first two are feature matrices for drugs/compounds and target kinases, respectively, and the last one is the drug-target interaction affinity matrix. For Davis, GPCR, IC, and E all pairwise interactions are known, whereas Metz, KiBA, and Merget are naturally sparse. That is, interaction information is available only for a small subset of all possible drug-target pairs. The numbers of drugs (n_D), targets (n_T), and their known interactions (n) are presented in Table 2. For more details of the data sets, we recommend the references provided in Table 2.

According to [39, 40, 45], we can divide pairwise prediction tasks into different experimental settings based on the assumptions regarding the overlap between the training data and the new pairs for which predictions are needed. The four distinct settings can be defined as

- $x \in S1$ if and only if $d \in X_d$ and $t \in X_t$
- $x \in S2$ if and only if $d \in X_d$ but $t \notin X_t$
- $x \in S3$ if and only if $d \notin X_d$ but $t \in X_t$

Table 2 Data set characteristics

Data set	n_D	n_T	n	Label type	References
Davis	68	442	30,056	Continuous	[9]
Metz	1421	156	93,356	Continuous	[33]
KIBA	2111	229	118,254	Continuous	[46]
Merget	2967	226	167,995	Continuous	[32]
G protein-coupled receptor (GPCR)	223	95	21,185	Binary	[51]
Ion Channels (IC)	210	204	42,840	Binary	[51]
Enzymes (E)	445	664	295,480	Binary	[51]

- $x \in S4$ if and only if $d \notin X_d$ and $t \notin X_t$.

Setting S1 corresponds to missing value imputation as both the drug and the target are encountered in the training set. Settings S2 and S3 correspond to multilabel learning problems, where the aim is to predict the labels for novel targets and drugs, respectively. Setting S4 is the most challenging setting called zero-shot learning. It corresponds to predicting labels for such pairs where neither the drug nor the target occurs in the training set. For more details about splitting the data in different settings, we refer to [39].

All data sets are divided five times to separate, randomly selected, training, validation, and test sets (1/3, 1/3, and 1/3) addressing the separate experimental settings accordingly. In what follows, we report the average result obtained under each setting in each data set. Naturally, the same sets are used with all learning methods.

7.2 Methods

The following pairwise kernel learning algorithms are used in our experiments:
CGKronRLS: We apply the state-of-the-art method `CGKronRLS` [1] as a benchmark for non-sparse solutions. `CGKronRLS` has demonstrated strong real-world performance, ranking among the top-performing methods in the IDG-DREAM Drug-Kinase Binding Prediction Challenge [7]. `CGKronRLS` uses the conjugate gradient (CG) method to solve the standard RLS formulation of the pairwise kernel learning problem (12). That is, it applies the squared loss $\|y - p\|^2$ as a loss function and the Euclidean norm $\frac{\lambda}{2}\|f\|^2_{\mathcal{H}}$ for regularization. The regularization parameter λ is selected from the grid $\{2^{-10}, 2^{-5}, 2^{-4}, 2^{-3}, 2^{-2}, 2^{-1}, 2^0, 2^1, 2^3, 2^4, 2^5, 2^{10}\}$. The tuning of λ is made separately for each data set and each setting. In `CGKronRLS`, the C-index of the validation data is used to select the best solution with an early stopping procedure [52].

`CGKronRLS` is implemented in Python/Cython and it belongs to RLScore [38] — the regularized least-squares machine learning algorithms package — available at https://github.com/aatapa/RLScore.

CGKronSVM: In addition, for binary labelled data sets, we apply `CGKronSVM` [1], a hinge loss variant of `CGKronRLS`, as hinge loss is generally considered more effective for classification than squared error loss. Notably, in its original introduction [1], `CGKronSVM` demonstrated slightly better predictive accuracy than `CGKronRLS` for binary-labeled data. `CGKronSVM` uses CG to solve the SVM formulation of the classification problem. It sometimes produces sparse solutions with an arbitrary level of sparsity. The parameters for `CGKronSVM` are selected similarly to `CGKronRLS`. In addition, `CGKronSVM` is imple-

mented in Python/Cython, belongs to RLScore, and is available at https://github.com/aatapa/RLScore.

LMBM-Kronℓ_0LS: LMBM-Kronℓ_0LS [37] is a nonsmooth optimization-based method designed for learning sparse models in pairwise interaction affinity prediction. This method is chosen as a benchmark because, like the proposed approach, it utilizes kernels and the continuous formulation of the ℓ_0-pseudonorm to address sparse pairwise learning problems. In LMBM-Kronℓ_0LS, the pairwise kernel learning problem (12) is formulated with squared loss and ℓ_0-pseudonorm regularization. The method employs LMBM to efficiently solve the resulting nonsmooth optimization problem. An early stopping procedure based on the C-index values of the validation data and the obtained sparsity level is applied.

Note that LMBM-Kronℓ_0LS is unable to find a solution with the predetermined sparsity percentage (cf. the proposed method). Instead, it aims to find a solution as sparse as possible while maintaining satisfactory prediction accuracy with respect to the non-sparse reference solution. This is done by solving a bi-objective optimization problem. Similar to [37], we allow a 5% decrease in C-index from the solution obtained with the reference method CGKronRLS. The main objective of the method is set to *MO1* and the starting point procedure *SP1* is used.[4] Otherwise, the parameters similar to [37] are used.

LMBM-Kronℓ_0LS is implemented in combination of Python and Fortran 95 via the F2PY interface. The source code of LMBM-Kronℓ_0LS is available at https://github.com/pavetsu14/LMBM-KronL0LS.

SparsePKL: SparsePKL is an implementation of Algorithm 2. We test the algorithm with every loss function described in Sect. 5.2. In what follows we call these different formulations

- DRSL—squared loss with double regularization;
- DRSEI—squared epsilon-intensive loss with double regularization;
- DREIS—epsilon-intensive squared loss with double regularization;
- DREIA—epsilon-intensive absolute loss with double regularization;
- DRAL—absolute loss with double regularization.

We employ the same set of parameters for all losses in penalty updating and optimization. The parameters are chosen to be generally effective across all losses in the Davis data set. Note that due to automatic definition of the (starting) regularization parameters ρ_1 and ρ_2^1 (see, Algorithm 2 Step 0) they are different for different data sets, settings, and loss functions. In addition, our preliminary experiments showed that DRAL and DREIA tend to produce trivial feasible solutions $a^* = 0$ in binary labeled data with the original starting point $a^1 = 1/n$ (see Algorithm 2, Step 0). Therefore, we used the starting point $a^1 = 1.0$ with these versions of SparsePKL in the binary labeled data sets GPCR, IC, and E.

There is no early stopping procedure employed in SparsePKL but the algorithm is terminated if a feasible critical point is found with the LMB- DCA or if the maximum number of iterations (50 in our experiments) in SparsePKL is reached. The best result is selected using the C-index of the validation data. For more details of the implementation, see Algorithm 2 and Sect. 6.2.

As already mentioned the source code of SparsePKL is available at https://github.com/napsu/sparsePKL.

All the learning algorithms use Kronecker product kernel matrices to compute the predictions $p = Ka$. The Kronecker product matrices are computed implicitly using the GVT [39,

[4] We only used one starting point procedure here due to the fact that even with one starting point procedure the method is the most time consuming of the methods tested and using also the second starting point procedure would have doubled the time.

48] from RLScore. The drug and target kernels are computed via Gaussian RBF kernels[5] with the kernel width parameter $\mu = 10^5$ as recommended in [1].

In addition to these pairwise kernel learning algorithms, we consider two DC optimization algorithms:

LMB-DCA: `LMB-DCA` is a Fortran 95 implementation of Algorithm 1. It is capable of solving any large-scale nonsmooth DC programming problem, although, here it is used as an underlying solver for pairwise kernel learning. The parameters are chosen to be generally effective across all losses in the Davis data set. The source code of `LMB-DCA` is included in `SparsePKL` and available at https://github.com/napsu/sparsePKL.

DCA: The `DCA` software applied here is a Fortran 95 implementation of the renowned DC programming algorithm DCA [27, 41]. We employ the LMBM to solve the underlying convex optimization problem at each iteration of `DCA`. The parameter selection used is similar to that of `LMB-DCA`. The source code of `DCA` is included in `SparsePKL` and available at https://github.com/napsu/sparsePKL.

Computational experiments are carried out on iMac, 4.0 GHz Intel(R) Core(TM) i7 machine with 16 GB of RAM. We use Python 3.7 and gfortran to compile the Fortran codes.

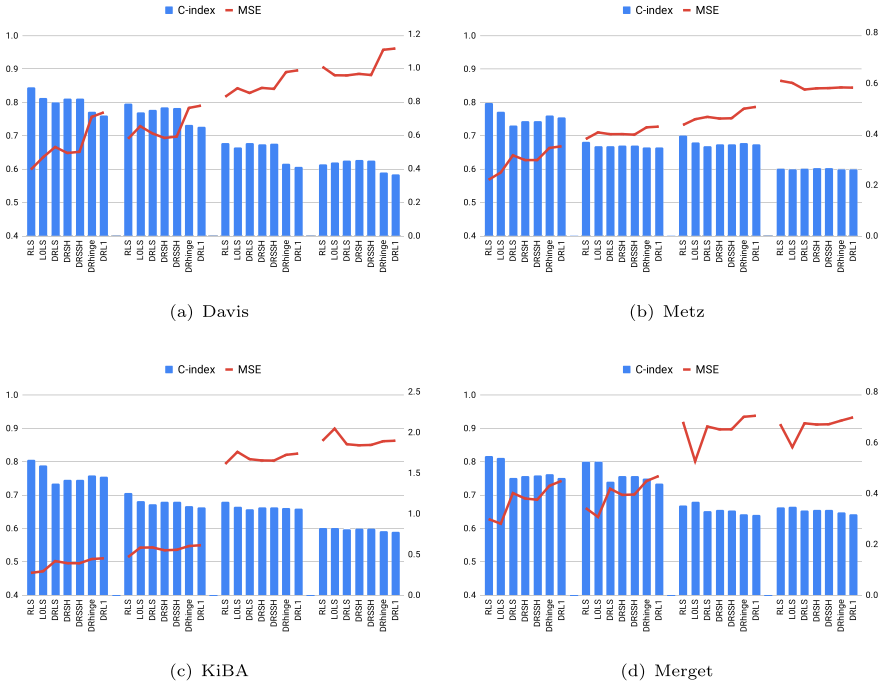
7.3 Results

The results of our numerical experiments are analyzed and evaluated using the C-index, mean squared error (MSE), sparsity of the solution, and the used CPU time. As already mentioned, the results are averaged over five different training, validation, and test set splits of each data. The C-index provides convenient performance metrics when it is more important to predict the relative order of labels than their exact values, while MSE measures these exact values. Note that any constant function obtains trivial 0.5 level performance with respect to the C-index. We say that the sparsity level is *zero* if the solution is dense (i.e. all variables are non-zero) and it is 100% if we have a trivial feasible solution $a^* = 0$. The CPU times for the algorithms are averaged over all runs in data. That is, they include computational times used in all different experimental settings S1–S4 in five different splits.

The results are given in Tables 4, 5, 6, 7, 8, 9 and10 in the appendix and summarized here in Figs. 2 and 3. We run the proposed `SparsePKL` with four desired levels of sparsity: 0% ($k = n$), 50% ($k = n/2$), 80% ($k = n/5$), and 90% ($k = n/10$). In Fig. 2 we give the C-indices (blue bar) and MSEs (red line) for different algorithms/loss functions in continuously labeled data sets. The desired sparsity level used with `SparsePKL` is 80% and it was always reached but once with `DREIS` in Davis data under setting S4. The results obtained with the other desired levels of sparsity are very similar but with the 90% sparsity, there were somewhat more infeasible solutions (see Tables 4, 5, 6 and 7). Note that `CGKronRLS` only produces dense solutions, while the final sparsity levels obtained with `LMBMKron`ℓ_0`LS` are given in the caption of the figure. In Fig. 3, we summarize the corresponding results (see Tables 8, 9 and 10) for binary labeled data. In addition, we report the results and sparsity levels obtained with `CGKronSVM`.

In Tables 4, 5, 6, 7, 8, 9 and10, we use the bold font to emphasize the best result (greatest C-index or smallest MSE) obtained with any loss function attached to `SparsePKL`. In addition, we use a blue pen to emphasize predictions and computational times that are better with `SparsePKL` with $k = n$ than those of `CGKronRLS` and we use a red pen to emphasize predictions that are better with `SparsePKL` than those of `LMBMKron`ℓ_0`LS`. In

[5] Our preliminary tests with other kernel combinations resulted in much worse predictions.



(a) Davis (b) Metz

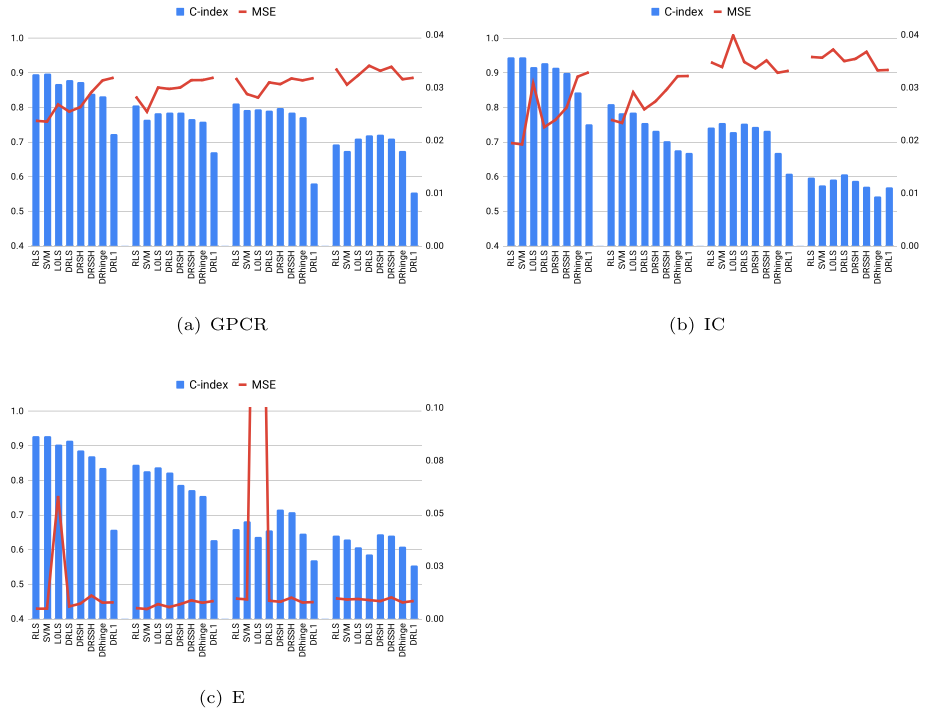
(c) KiBA (d) Merget

Fig. 2 C-index and MSE of pairwise kernel learning algorithms in data with continuous labels under different settings (S1 in leftmost and S4 in rightmost abscissa). The desired sparsity level used with `SparsePKL` is 80%. The sparsity level obtained with `CGKronRLS` (RLS in figures) is always about 0.0%. The sparsity levels obtained with `LMBMKron`ℓ_0`LS` (L0LS in figures) under settings S1, S2, S3, and S4 are **a** 73.60%, 90.54%, 90.51%, and 90.59%; **b** 54.63%, 82.38%, 74.44%, and 90.73%; **c** 59.36%, 71.21%, 88.58%, and 91.68%; **d** 34.62%, 46.58%, 57.73%, and 78.39%, respectively

the latter, we only compare the results which have approximately the same sparsity levels. That is, the results we select for `SparsePKL` depend on the sparsity levels reached with `LMBMKron`ℓ_0`LS`. If none of the solutions is better, we use the blue (red) pen to the result obtained with `CGKronRLS` (`LMBMKron`ℓ_0`LS`). Further, in the tables, $nz\%$ denotes the percentages of the variables that are nonzero at the solution (i.e. it is opposite to the sparsity level) and the CPU time is given in seconds.

7.3.1 General observations

As expected, the highest prediction accuracy was obtained under the most informative setting S1 while the more realistic settings S2, S3, and S4 resulted in reduced accuracies (see Figs. 2 and 3). Setting S2 showed often higher accuracy than S3 suggesting that predicting new targets for drugs is easier than predicting new targeted compounds. Moreover, predicting labels for unseen drugs and targets (S4, zero-shot learning) was clearly the most challenging task for all methods. Nevertheless, also in this case, a predicting function was successfully learned in the majority of the data sets. Further, we note that learning to predict is usually easier in the binary labeled data sets GPCR, IC, and E than in the continuously labeled data sets Davis, Metz, KiBA, and Merget. This result is consistent with the one obtained in [39].



(a) GPCR (b) IC

(c) E

Fig. 3 C-index and MSE of pairwise kernel learning algorithms in data with binary labels under different settings (S1 in leftmost and S4 in rightmost abscissa). The desired sparsity level used with `SparsePKL` is 80%. The sparsity level obtained with `CGKronRLS` (RLS in figures) is always about 0.0%. The sparsity levels obtained with `CGKronSVM` (SVM in figures) and `LMBMKron`ℓ_0`LS` (L0LS in figures) under settings S1, S2, S3, and S4 are **a** `CGKronSVM`: 0.05%, 19.55%, 1.46%, and 76.21%, `LMBMKron`ℓ_0`LS`: 89.63%, 87.81%, 92.68%, and 88.21%; **b** `CGKronSVM`: 0.07%, 0.09%, 0.10%, and 0.48%, `LMBMKron`ℓ_0`LS`: 87.00%, 51.81%, 89.48%, and 72.26%; **c** `CGKronSVM`: 0.24%, 0.16%, 0.81%, and 19.86%, `LMBMKron`ℓ_0`LS`: 92.00%, 90.22%, 86.64% and 64.62%

One might expect that increasing the desired sparsity level would decrease the obtained prediction accuracy. Indeed, this is usually the case under settings S1–S3 in continuously labeled data sets (see Tables 4, 5, 6 and 7). However, in binary labeled data sets (see Tables 8–10) and/or in zero-shot learning (regardless of the type of the labels) the obtained predictions with enforced sparsity are often more accurate than the corresponding predictions produced with dense solution vectors. In the first case, this is just due to the characteristics of the data. In zero-shot learning, there is a lot of uncertainty involved due to the fact that neither the drug nor the target has been seen in the training phase. Employing sparse solution vectors enables us to recognize the most essential drug-target interactions, helping us uncover the real patterns within the data while disregarding the redundant ones. This, in turn, leads to improved prediction accuracy.

7.3.2 Comparison with `CGKronRLS`, `CGKronSVM`, and `LMBMKron`$\boldsymbol{\ell_0}$`LS`

A quick look at Tables 4, 5, 6, 7, 8, 9 and 10 (see the blue pen) reveals that `CGKronRLS` is always the most accurate predictor under the simplest setting S1 and also often in settings S2 and S3 when no sparsity is required. On the other hand, in setting S4, there is always a

version of SparsePKL that produces more accurate predictions than CGKronRLS[6]. As noted before, the sparsity of the solution vector may give SparsePKL some advantage in setting S4. However, the above result holds true (but in one data) even if we only compare the SparsePKL results with $k = n$. The reason for improved prediction accuracy is most probably the usage of the nonsmooth optimization solver LMB-DCA that is less sensitive to all kinds of perturbations than the smooth CG solver. The CPU times used with DRSEI and DREIS are always significantly smaller than those of CGKronRLS. On the other hand, CGKronRLS clearly win the other loss functions applied within SparsePKL.

The comparison between LMBMKronℓ_0LS and SparsePKL (see the red pen in Tables 4, 5, 6, 7, 8, 9 and 10) reveals that LMBMKronℓ_0LS often finds sparse solutions with a bit more accurate predictions. More so, if we only compare the results in continuously labeled data with respect to the C-index. However, in binary labeled data and/or with respect to MSE different versions of SparsePKL start to look attractive alternatives. It is also worth noting that most results obtained with SparsePKL are still pretty accurate predictions and the used computational times are vastly shorter than those of LMBMKronℓ_0LS: for instance, in Merget DREIS with $k = n/2$ is about 150 times faster than LMBMKronℓ_0LS. Besides, the implementation of LMBMKronℓ_0LS seems to be a bit unstable and the method crashed often with an unknown reason, especially with bigger data sets (the results in tables are averaged over the successful runs).

In binary labeled data sets CGKronSVM is always the best algorithm with respect to both the C-index and MSE under setting S1 (sometimes together with CGKronRLS, see Fig. 3). However, this result does not hold with more demanding settings S2–S4. In addition, the solutions provided by CGKronSVM are usually not very sparse and they take longer to compute than those of CGKronRLS and most versions of SparsePKL (see Tables 8–10).

Hence, we conclude that CGKronRLS is a very good predictor if the prediction task falls under simple settings S1–S3 and no sparsity of the solution is required. Otherwise, particularly for zero-shot learning, using SparsePKL with a suitable loss function is a preferable option.

7.3.3 Different loss functions

Now we compare different loss functions used with SparsePKL. First, we point out that the nonlinear losses DRSL, DRSEI, and DREIS usually produce more accurate predictions than the piecewise linear losses DRAL and DREIA. An exception here is DREIA under setting S1. In addition, DRSEI and DREIS find these predictions efficiently.

The original LMBM is shown to be best suited for solving highly nonlinear (and nonsmooth) optimization problems [25]. That is, it works better — both in terms of accuracy and efficiency — in nonlinear than in (piecewise) linear cases. As a successor of the LMBM the proposed LMB-DCA most probably shares this property, which explains the previous result. The inefficiency of DRSL is because it never triggers the termination criterion of the LMB-DCA (see Step 3 of Algorithm 1). Instead, it terminates only when the maximum number of iterations in the LMB-DCA is reached. This also means that it runs until the maximum number of iterations in the SparsePKL algorithm (see Algorithm 2) is reached. Here, an easy fix would be an early stopping procedure using the validation data.

The predictions produced by DRAL are often clearly less accurate than those of the others. The same is true for DREIA but with less clear deterioration. There are two possible reasons

[6] To be precise, in IC data under S4, CGKronRLS produces the most accurate prediction w.r.t. C-index if we only compare it to SparsePKL results with $k = n$. However, both DRSL and DRSEI predict better than CGKronRLS with higher sparsity levels (see Table 9).

for this. The first one is the used optimization method LMB- DCA as both absolute loss and epsilon-intensive absolute loss comprise only piecewise linear terms. The second one is the used starting point. As already mentioned, DRAL and DREIA tend to produce trivial feasible solutions $a^* = 0$ in binary labeled data when the starting point is too small. Although trivial feasible solutions are not obtained in continuously labeled data, bigger starting points might work better (give more accurate predictions) with these loss functions in continuously labeled data as well but then the number of iterations should be increased in order to get feasible solutions as the convergence to feasible solutions with these two losses is quite slow.

As a result, we recommend to use SparsePKL with a nonlinear (nonsmooth) loss. More precisely, apply DRSL under settings S1 and either DRSEI or DREIS with settings S2–S4.

7.3.4 LMB-DCA vs. DCA

Finally, we compare LMB-DCA with the well-known DCA to show its performance as a pure optimization method. The DCA applied here employs the LMBM as an underlying solver. Therefore, the comparison should be quite fair as both solvers are capable of solving large-scale nonsmooth optimization problems and internal parameter selection is similar. We use the same data sets as before, with one split to different experimental settings each, and the epsilon-intensive squared loss as an objective function. The desired sparsity level is set to 50% (if we set $k = n$ we would have a convex problem) but only one fixed value 0.0001 is used for both ρ_1 and ρ_2. In practice, this means that we cannot expect to get feasible solutions in terms of sparsity level. However, we can compare the values of the objectives at the end of the optimization procedure (the training procedure without intermediate validation).

In Table 3 we report

- n—the number of variables in optimization: in addition to data, this value depends on the experimental setting we are using;
- J^*—the obtained value of the objective function;
- n_{J1}, n_{J2}—numbers of DC component evaluations used with DCA;
- $n_{\xi 1}$, $n_{\xi 2}$—numbers of subgradient evaluations for the DC components used with DCA;
- n_J and n_ξ—numbers of function and subgradient evaluations for LMB-DCA: here $n_J = n_{J1} = n_{J2}$ and $n_\xi = n_{\xi 1} = n_{\xi 2}$, that is both DC components (subgradients) were evaluated n_J (n_ξ) times[7];
- CPU—the used CPU times in seconds.

In the table, bold font is used for the better objective value and for the better CPU time. It is easy to see that LMB-DCA outperforms DCA in terms of efficiency. The differences in accuracy are not as significant. However, there is a clear difference in the sense that LMB-DCA found slightly smaller objective values in data sets with continuous labels while DCA found smaller values in data with binary labels (see Table 2). In addition, if we solved a DC optimization problem with a computationally more expensive second DC component J_2, DCA would be an obvious choice as it uses only very few evaluations of the second DC component.

8 Conclusions

In this paper, we have introduced the *sparse pairwise kernel learning algorithm* (SparsePKL) that produces accurate predictions with the desired sparsity level of the solution. The most

[7] This is due to the fact that the DCA step (Step 9 in Algorithm 1) was not applied in these experiments. Nevertheless, it was employed semi-regularly in the experiments of the previous sections.

Table 3 Comparison of LMB-DCA and DCA

Data	Setting	n	LMB-DCA				DCA					
			J^*	n_J	n_ξ	CPU	J^*	n_{J1}	n_{J2}	$n_{\xi 1}$	$n_{\xi 2}$	CPU
Davis	S1	6638	**0.3057**	1428	1426	13.56	0.3082	1120	7	1075	7	**12.59**
	S2	3234	0.2881	956	956	**2.31**	**0.2877**	1696	10	1617	10	5.06
	S3	3404	0.2631	317	317	**0.90**	**0.2575**	1647	8	1600	8	5.20
	S4	3381	**0.3142**	1512	1510	**3.76**	0.3149	1195	9	1111	9	4.00
Metz	S1	20, 307	**0.2208**	1363	1358	**86.68**	0.2229	1105	6	1053	6	99.43
	S2	10, 044	**0.2070**	1656	1649	**20.65**	0.2078	1920	14	1741	14	33.55
	S3	10, 967	**0.1755**	1384	1377	**19.12**	0.1756	1831	14	1646	14	35.34
	S4	11, 487	**0.2039**	1534	1531	**22.16**	0.2045	1758	9	1669	9	35.84
KiBA	S1	26, 593	**0.3324**	2064	2061	**267.18**	0.3336	2268	10	2160	10	398.74
	S2	14, 177	**0.2708**	2097	2094	**52.72**	0.2731	2197	16	1930	16	75.28
	S3	20, 315	**0.2394**	1804	1802	**64.61**	0.2407	2392	16	2170	16	120.08
	S4	20, 909	**0.2345**	1897	1895	**22.16**	0.2369	1891	8	1814	8	98.34
Merget	S1	37, 085	**0.2767**	1724	1715	**426.11**	0.2770	1834	8	1734	8	587.63
	S2	19, 728	**0.2754**	1618	1614	82.65	0.2766	1136	7	1067	7	**71.59**
	S3	20, 435	**0.2397**	990	985	**50.54**	0.2398	1426	14	1234	14	89.58
	S4	23, 202	**0.2478**	928	924	**54.88**	0.2482	1583	20	1226	20	114.84
GPCR	S1	4715	0.0139	254	252	**1.41**	**0.0138**	470	8	380	8	2.94
	S2	2294	0.0195	154	154	**0.40**	**0.0194**	387	9	289	9	0.74
	S3	2368	0.0125	159	159	**0.43**	**0.0124**	415	9	317	9	0.83
	S4	2294	0.0133	207	206	**0.48**	**0.0132**	406	7	330	7	0.77
IC	S1	9486	0.0131	361	357	**4.54**	**0.0129**	604	5	546	5	9.08
	S2	4692	**0.0098**	242	241	**0.95**	**0.0098**	903	22	528	22	3.56
	S3	4760	**0.0138**	218	212	**0.87**	**0.0138**	416	6	338	6	1.87
	S4	4760	**0.0153**	291	286	**1.13**	0.0155	429	6	375	6	1.95
E	S1	65, 675	0.0052	951	905	**175.04**	**0.0048**	1408	10	1259	10	333.40
	S2	32, 708	0.0049	525	515	**20.30**	**0.0039**	936	11	761	11	55.08
	S3	32, 856	0.0054	723	710	**28.24**	**0.0051**	1493	17	1212	17	86.01
	S4	32, 708	0.0056	842	825	**32.57**	**0.0055**	1513	18	1259	18	87.55

important reason to seek a sparse solution vector—that is, a vector with only a few nonzero entries—is that it enables us to recognize the most essential input pairs and omit the redundant ones. This leads to improved prediction accuracy, especially under the most challenging experimental setting called zero-shot learning. Moreover, having more zeros in the solution vector speeds up the prediction process. This is due to the fact that in the prediction phase, the dominating costs are the kernel matrix multiplications with the sparse dual coefficient vector.

In addition to the SPARSEPKL, we have proposed a novel *limited memory bundle DC algorithm* (LMB- DCA) for large-scale nonsmooth DC optimization and used it as an underlying solver in the SPARSEPKL. As a pure optimization method, the proposed LMB- DCA outperforms the well-known DCA in terms of efficiency (unless we would solve a DC optimization problem with a computationally very expensive second DC component) while the accuracies of these algorithms are similar.

We have evaluated the performance of the SPARSEPKL with various loss functions in seven real-life drug-target interaction data sets. The data sets are divided into different experimental settings S1–S4 (including zero-shot learning) according to [39] and different levels of sparsity are tested with the SPARSEPKL. As a result, we recommend to use `SparsePKL` with a nonlinear (nonsmooth) loss. More precisely, apply the squared loss under settings S1 (may be used with S2 and S3 in binary labeled data as well) and either the squared epsilon-intensive loss or epsilon-intensive squared loss with settings S2–S4.

The obtained results are compared against the state-of-the-art methods KronRLS and KronSVM [1, 38], as well as the LMBM-Kronℓ_0LS algorithm [37] that produces as sparse a solution as possible while keeping the predetermined accuracy of the prediction. The proposed algorithm outperformed the LMBM-Kronℓ_0LS by producing relatively accurate predictions within only a small fraction of the time used by the LMBM-Kronℓ_0LS. In addition, the proposed algorithm produced more accurate predictions than KronRLS and KronSVM under the most challenging setting S4. It usually also made it faster.

We conclude that the KronRLS is a very good predictor if the prediction task falls under the simple settings S1–S3 and no sparsity of the solution is required. Otherwise, particularly for zero-shot learning, using the SPARSEPKL with a suitable loss function is a preferable option.

Appendix A: Detailed numerical results

The results of our numerical experiments are given in Tables 4, 5, 6, 7, 8, 9 and 10. We recall that the results are averaged over five different training, validation, and test splits of each data.

In the tables, we use the bold font to emphasize the best result (greatest C-index or smallest MSE) obtained with any loss function attached to `SparsePKL`. We use a blue pen to emphasize predictions and computational times that are better or equal with `SparsePKL` with $k = n$ than those of `CGKronRLS`. If none of the solutions is better, we use the blue pen to the result obtained with `CGKronRLS`. In addition, we use a red pen to emphasize predictions of `SparsePKL` that are more or equally accurate than those obtained with `LMBMKron`ℓ_0`LS`.

Table 4 Results in Davis

Method/	k	S1			S2			S3			S4			CPU
Objective		CI	MSE	nz%	CI	MSE	nz%	CI	MSE	nz%	CI	MSE	nz%	
`CGKronRLS`	n	0.845	0.394	100.00	0.797	0.577	100.00	0.678	0.827	100.00	0.614	1.005	100.00	9.76
`LMBMKron`ℓ_0`LS`	-	0.814	0.467	26.40	0.770	0.652	9.46	0.665	0.878	9.49	0.620	0.955	9.41	323.10
`SparsePKL/`	n	**0.834**	**0.417**	99.98	**0.795**	0.573	99.98	0.675	0.857	99.99	0.628	0.978	99.98	127.09
`DRSL`	$n/2$	0.809	0.511	49.78	0.785	0.588	49.67	**0.681**	**0.815**	49.69	0.635	0.957	49.75	13.89
	$n/5$	0.801	0.528	19.94	0.778	0.609	19.98	**0.678**	**0.850**	19.89	0.626	**0.954**	19.95	40.90
	$n/10$	0.783	0.564	9.96	0.747	0.671	9.94	**0.659**	**0.932**	9.94	0.604	**0.951**	9.96	78.51
`SparsePKL/`	n	0.823	0.466	99.99	0.793	**0.558**	99.98	0.675	0.828	99.99	0.630	0.970	100.00	7.33
`DRSEI`	$n/2$	**0.819**	**0.479**	49.68	**0.792**	**0.557**	49.73	0.677	0.827	49.77	0.633	0.968	49.78	13.78
	$n/5$	**0.812**	**0.492**	19.96	**0.785**	**0.582**	19.96	0.674	0.880	19.94	**0.627**	0.963	19.98	30.38
	$n/10$	0.794	0.541	9.97	**0.763**	0.655	9.98	0.650	0.958	9.94	0.601	1.078	9.99	64.96
`SparsePKL/`	n	0.818	0.483	100.00	0.792	0.565	99.98	**0.684**	**0.823**	99.96	**0.635**	**0.958**	99.99	**6.37**
`DREIS`[1]	$n/2$	0.817	0.488	49.83	0.790	0.569	49.67	0.679	0.823	49.72	**0.636**	**0.957**	49.74	**11.35**
	$n/5$	**0.812**	0.500	19.96	0.784	0.590	19.97	0.677	0.874	19.92	0.625	0.956	19.97	**24.93**
	$n/10$	**0.798**	**0.524**	9.97	0.760	**0.644**	9.94	0.648	0.934	9.96	**0.610**	0.961	9.99	**45.01**
`SparsePKL/`	n	0.790	0.656	99.95	0.747	0.716	99.93	0.627	0.969	99.95	0.597	1.099	99.89	27.31
`DREIA`	$n/2$	0.781	0.686	49.69	0.742	0.735	49.78	0.623	0.966	49.67	0.594	1.097	49.68	17.66
	$n/5$	0.771	0.707	19.97	0.732	0.761	19.96	0.615	0.974	19.96	0.590	1.107	19.89	40.07
	$n/10$	0.753	0.731	9.98	0.717	0.801	9.97	0.597	1.005	9.95	0.585	1.121	9.99	59.91
`SparsePKL/`	n	0.785	0.673	99.94	0.741	0.744	99.96	0.624	0.983	99.86	0.592	1.107	99.91	40.27
`DRAL`	$n/2$	0.777	0.699	49.70	0.737	0.749	49.66	0.620	0.971	49.68	0.590	1.106	49.62	17.93
	$n/5$	0.761	0.734	19.97	0.728	0.774	19.93	0.606	0.985	19.94	0.584	1.114	19.93	39.37
	$n/10$	0.747	0.760	9.97	0.711	0.821	9.98	0.592	1.013	9.97	0.576	1.128	9.96	59.84

[1] `DREIS` with $k = n/20$: 1/5 infeasible solution with S4 (nz% $\approx$ 20.05)

Table 5 Results in Metz

Method/	k	S1			S2			S3			S4			CPU
Objective		CI	MSE	nz%	CI	MSE	nz%	CI	MSE	nz%	CI	MSE	nz%	
CGKronRLS	n	0.799	0.221	100.00	0.681	0.381	100.00	0.700	0.436	100.00	0.601	0.610	99.87	180.18
LMBMKronℓ_0LS[1]	-	0.772	0.249	45.37	0.669	0.407	17.62	0.681	0.459	25.56	0.600	0.601	9.27	8692.19
SparsePKL/ DRSL[2]	n	0.787	**0.242**	99.99	**0.678**	**0.393**	99.97	**0.697**	0.451	100.00	**0.602**	0.589	99.97	981.20
	$n/2$	0.746	0.302	49.65	0.673	0.396	49.68	0.680	0.464	49.72	0.602	**0.578**	49.88	105.72
	$n/5$	0.731	0.317	19.98	0.668	0.400	19.98	0.668	0.468	19.97	**0.602**	**0.575**	19.96	255.39
	$n/10$	0.710	0.340	9.96	0.654	**0.424**	10.06	0.648	0.494	9.98	0.601	0.590	9.98	704.63
SparsePKL/ DRSEI	n	0.769	0.271	99.96	0.675	**0.393**	99.97	0.690	0.451	99.98	0.600	0.589	99.98	80.79
	$n/2$	0.759	**0.285**	49.67	0.673	0.396	49.72	0.685	0.455	49.79	**0.603**	0.581	49.86	94.98
	$n/5$	0.743	**0.298**	19.96	**0.670**	0.400	19.99	0.674	**0.461**	19.97	**0.602**	0.580	19.95	246.77
	$n/10$	0.718	0.327	9.99	0.656	**0.424**	9.98	0.656	**0.474**	9.98	**0.604**	**0.578**	9.98	429.63
SparsePKL/ DREIS	n	0.771	0.268	99.97	0.675	0.394	99.98	0.691	**0.447**	99.98	0.601	**0.584**	99.97	**76.36**
	$n/2$	0.759	**0.285**	49.67	**0.674**	**0.395**	49.77	**0.686**	**0.452**	49.84	**0.603**	0.583	49.71	**76.54**
	$n/5$	0.744	**0.298**	19.94	**0.670**	**0.398**	19.98	0.674	0.462	19.97	**0.602**	0.581	19.97	**211.97**
	$n/10$	0.720	**0.325**	9.99	0.657	**0.424**	9.98	0.657	**0.474**	9.98	0.603	0.585	9.98	**323.63**
SparsePKL/ DREIA[3]	n	**0.788**	0.308	99.97	0.670	0.420	99.94	0.695	0.480	99.97	0.601	**0.584**	99.94	962.02
	$n/2$	**0.768**	0.337	49.77	0.667	0.423	49.77	0.685	0.498	49.79	0.599	0.590	49.67	139.93
	$n/5$	**0.760**	0.345	19.96	0.666	0.427	19.95	**0.679**	0.500	19.96	0.599	0.584	19.98	247.07
	$n/10$	**0.751**	0.354	9.99	**0.661**	0.437	10.00	**0.665**	0.513	10.00	0.597	0.588	10.03	423.37
SparsePKL/ DRAL[4]	n	0.782	0.319	99.95	0.668	0.424	99.93	0.692	0.488	99.96	0.599	0.594	99.95	927.92
	$n/2$	0.763	0.345	49.66	0.666	0.428	49.82	0.682	0.503	49.83	0.599	0.589	49.63	145.85
	$n/5$	0.756	0.352	19.95	0.666	0.429	19.93	0.674	0.507	19.95	0.599	0.583	19.95	241.09
	$n/10$	0.742	0.367	10.06	0.660	0.443	10.01	0.664	0.515	10.05	0.598	0.580	10.00	435.64

[1] LMBMKronℓ_0LS results are averaged over four successful runs
[2] DRSL with $k = n/10$: 1/5 infeasible solution with S2 (nz% $\approx$ 10.45) and S3 (nz% $\approx$ 10.01)
[3] DREIA with $k = n/10$: 1/5 infeasible solution with S1 (nz% $\approx$ 10.01) and 2/5 infeasible solutions with S3 (nz% $<$ 10.09) and S4 (nz% $<$ 10.15)
[4] DRAL with $k = n/10$: 4/5 infeasible solutions with S1 (nz% $<$ 10.14) and S3 (nz% $<$ 10.12), and 1/5 infeasible solution with S2 (nz% $\approx$ 10.07) and S4 (nz% $\approx$ 10.07)

Table 6 Results in KiBA

Method/	k	S1			S2			S3			S4			CPU
Objective		CI	MSE	nz%	CI	MSE	nz%	CI	MSE	nz%	CI	MSE	nz%	
CGKronRLS	n	0.805	0.273	100.00	0.706	0.468	100.00	0.680	1.612	100.00	0.602	1.897	99.97	325.23
LMBMKronℓ_0LS[1]	-	0.789	0.293	40.64	0.682	0.583	28.79	0.665	1.760	11.42	0.601	2.046	8.32	13859.78
SparsePKL/ DRSL[2]	n	**0.796**	**0.293**	99.96	**0.704**	**0.471**	99.92	**0.679**	**1.601**	99.98	**0.605**	1.809	99.97	2053.47
	$n/2$	0.753	0.394	49.73	0.685	0.529	49.73	0.668	1.650	49.83	0.601	1.831	49.75	224.24
	$n/5$	0.735	0.418	19.97	0.672	0.586	19.94	0.658	1.671	19.97	0.597	1.855	19.95	626.91
	$n/10$	0.699	0.488	10.02	0.650	0.669	10.00	0.643	1.730	9.95	0.591	1.919	9.96	1601.39
SparsePKL/ DRSEI[3]	n	0.783	0.324	99.98	0.697	0.491	99.98	0.672	1.632	99.97	**0.605**	**1.806**	99.98	**250.32**
	$n/2$	0.766	**0.363**	49.69	0.689	0.515	49.73	**0.673**	**1.628**	49.78	**0.603**	**1.816**	49.78	205.45
	$n/5$	0.745	0.392	19.98	**0.681**	**0.551**	19.98	0.663	1.655	19.97	**0.600**	**1.841**	19.96	461.21
	$n/10$	0.715	**0.447**	10.01	**0.659**	0.651	9.97	0.647	**1.708**	10.00	**0.594**	**1.896**	9.97	**857.50**
SparsePKL/ DREIS[4]	n	0.783	0.325	99.98	0.696	0.492	99.98	0.676	1.620	99.96	0.604	1.813	99.97	275.64
	$n/2$	0.765	0.364	49.75	**0.690**	**0.512**	49.73	**0.673**	**1.628**	49.80	**0.603**	**1.816**	49.79	**199.01**
	$n/5$	0.746	**0.391**	19.97	0.680	0.557	19.94	**0.664**	**1.653**	19.95	0.599	1.846	19.98	**429.20**
	$n/10$	0.714	0.451	9.98	**0.659**	0.661	9.98	0.646	1.714	9.99	**0.594**	1.899	10.01	885.94
SparsePKL/ DREIA[5]	n	0.791	0.381	99.98	0.686	0.541	99.99	0.674	1.661	99.97	0.596	**1.850**	99.96	2016.55
	$n/2$	**0.771**	0.429	49.66	0.673	0.587	49.79	0.665	1.715	49.70	0.592	1.887	49.77	281.90
	$n/5$	**0.760**	0.443	19.97	0.667	0.603	19.98	0.661	1.724	19.94	0.591	1.891	19.98	582.28
	$n/10$	**0.741**	0.469	10.05	0.654	**0.646**	9.99	**0.652**	1.750	10.05	0.589	1.913	10.12	1016.06
SparsePKL/ DRAL[6]	n	0.784	0.400	99.98	0.679	0.565	99.97	0.671	1.681	99.97	0.593	1.911	99.95	2021.17
	$n/2$	0.766	0.439	49.67	0.669	0.599	49.83	0.663	1.732	49.70	0.590	1.900	49.77	285.03
	$n/5$	0.755	0.453	19.96	0.664	0.613	19.95	0.658	1.740	19.97	0.590	1.898	19.95	599.37
	$n/10$	0.737	0.476	10.10	0.651	0.652	9.98	0.651	1.757	10.09	0.588	1.917	10.05	1015.58

[1] LMBMKronℓ_0LS results are averaged over two successful runs
[2] DRSL with $k = n/10$: 2/5 infeasible solutions with S1 (nz% $<$ 10.12) and 1/5 infeasible solution with S2 (nz% $\approx$ 10.12) and S4 (nz% $\approx$ 10.01)
[3] DRSEI with $k = n/10$: 1/5 infeasible solution with S1 (nz% $\approx$ 10.15) and S3 (nz% $\approx$ 10.10)
[4] DREIS with $k = n/10$: 1/5 infeasible solution with S4 (nz% $\approx$ 10.13)
[5] DREIA with $k = n/10$: 4/5 infeasible solutions with S1 (nz% $<$ 10.10) and S3 (nz% $<$ 10.07), 1/5 infeasible solution with S2 (nz% $\approx$ 10.05), and 5/5 infeasible solutions with S4 (nz% $<$ 10.22)
[6] DRAL with $k = n/10$: 5/5 infeasible solutions with S1 (nz% $<$ 10.19), 1/5 infeasible solution with S2 (nz% $\approx$ 10.06), 4/5 infeasible solutions with S3 (nz% $<$ 10.25), and 3/5 infeasible solutions with S4 (nz% $<$ 10.27)

Table 7 Results in Merget

Method/	k	S1			S2			S3			S4			CPU
Objective		CI	MSE	nz%	CI	MSE	nz%	CI	MSE	nz%	CI	MSE	nz%	
CGKronRLS	n	0.817	0.299	100.00	0.800	0.342	100.00	0.669	0.681	100.00	0.662	0.672	100.00	482.89
LMBMKronℓ_0LS[1]	-	0.811	0.281	65.38	0.801	0.308	53.42	0.680	0.527	42.27	0.665	0.582	21.61	52194.98
SparsePKL/	n	**0.807**	**0.321**	99.96	**0.798**	**0.349**	99.95	0.660	0.667	99.91	**0.664**	**0.650**	99.90	3520.84
DRSL[2]	$n/2$	0.787	0.359	49.79	0.782	0.375	49.81	0.660	**0.638**	49.78	0.661	**0.652**	49.83	517.85
	$n/5$	0.752	0.401	19.95	0.739	0.418	19.94	0.652	0.663	19.96	0.653	0.676	19.94	1998.83
	$n/10$	0.728	0.446	10.06	0.695	0.502	9.99	0.638	0.707	9.95	**0.640**	0.710	9.97	2800.59
SparsePKL/	n	0.800	0.337	99.90	0.792	0.361	99.92	**0.661**	**0.635**	99.88	0.661	0.653	99.89	228.29
DRSEI[3]	$n/2$	**0.796**	0.342	49.91	**0.790**	0.363	49.77	**0.662**	**0.638**	49.75	**0.663**	0.653	49.78	405.16
	$n/5$	0.756	0.380	19.97	0.757	0.394	19.98	**0.656**	**0.652**	19.96	**0.655**	**0.671**	19.95	1332.48
	$n/10$	0.722	**0.441**	10.08	**0.715**	0.450	9.98	**0.642**	**0.678**	9.98	0.637	**0.700**	10.01	1971.45
SparsePKL/	n	0.800	0.337	99.91	0.792	0.361	99.90	**0.661**	0.638	99.89	0.661	0.654	99.88	**187.42**
DREIS[4]	$n/2$	**0.796**	**0.341**	49.80	**0.790**	**0.362**	49.79	0.661	**0.638**	49.79	0.662	0.654	49.86	**338.44**
	$n/5$	0.758	**0.375**	19.96	**0.758**	**0.396**	19.98	0.654	**0.652**	19.99	**0.655**	0.672	19.98	**1127.05**
	$n/10$	0.722	0.444	10.06	**0.715**	**0.447**	10.03	0.640	0.687	9.98	0.636	0.701	10.00	**1475.31**
SparsePKL/	n	0.796	0.388	99.87	0.793	0.393	99.87	0.655	0.672	99.88	**0.664**	0.667	99.86	3448.84
DREIA[5,6]	$n/2$	0.782	0.412	49.68	0.778	0.421	49.71	0.652	0.684	49.80	0.658	0.673	49.85	927.93
	$n/5$	**0.763**	0.430	20.03	0.749	0.449	19.96	0.643	0.701	19.98	0.648	0.686	19.98	2037.18
	$n/10$	**0.731**	0.472	10.09	0.702	0.531	10.08	0.634	0.730	10.05	0.631	0.724	10.07	2553.45
SparsePKL/	n	0.787	0.406	99.83	0.787	0.406	99.86	0.654	0.679	99.81	0.660	0.676	99.84	3335.65
DRAL[7]	$n/2$	0.773	0.430	49.69	0.770	0.435	49.77	0.649	0.691	49.72	0.654	0.683	49.82	970.84
	$n/5$	0.752	0.449	20.02	0.734	0.468	19.99	0.641	0.706	19.95	0.643	0.699	19.99	1922.67
	$n/10$	0.717	0.504	10.29	0.693	0.539	10.06	0.630	0.734	10.02	0.633	0.738	10.02	2321.60

[1] LMBMKronℓ_0LS results are obtained from only one successful run

[2] DRSL with $k = n/10$: 2/5 infeasible solutions with S1 (nz% $<$ 10.19) and 1/5 infeasible solution with S2 (nz% $\approx$ 10.02).

[3] DRSEI with $k = n/10$: 5/5 infeasible solutions with S1 (nz% $<$ 10.18) and 1/5 infeasible solution with S4 (nz% $\approx$ 10.14).

[4] DREIS with $k = n/10$: 5/5 infeasible solutions with S1 (nz% $<$ 10.10), 3/5 infeasible solutions with S2 (nz% $<$ 10.12) and 1/5 infeasible solution with S4 (nz% $\approx$ 10.09).

[5] DREIA with $k = n/5$: 3/5 infeasible solutions with S1 (nz% $<$ 20.16).

[6] DREIA with $k = n/10$: 5/5 infeasible solutions with S1 (nz% $<$ 10.15) and S4 (nz% $<$ 10.23), and 4/5 infeasible solutions with S2 (nz% $<$ 10.17) and S3 (nz% $<$ 10.09).

[7] DRAL with $k = n/10$: 5/5 infeasible solutions with S1 (nz% $<$ 10.53), 3/5 infeasible solutions with S2 (nz% $<$ 10.26) and S4 (nz% $\approx$ 10.08), and 4/5 infeasible solutions with S3 (nz% $<$ 10.08)

Table 8 Results in GPCR

Method/	k	S1			S2			S3			S4			CPU
Objective		CI	MSE	nz%	CI	MSE	nz%	CI	MSE	nz%	CI	MSE	nz%	
CGKronRLS	n	0.896	0.024	99.93	0.805	0.028	99.33	0.811	0.032	99.70	0.694	0.034	81.02	4.35
CGKronSVM	n	0.899	0.024	99.95	0.765	0.025	80.45	0.792	0.029	98.54	0.675	0.031	23.79	20.13
LMBMKronℓ_0LS[1]	-	0.868	0.027	10.37	0.783	0.030	12.19	0.794	0.028	7.32	0.711	0.032	11.79	93.37
SparsePKL/	n	**0.888**	**0.024**	99.96	**0.801**	**0.026**	99.98	**0.809**	0.032	99.92	0.713	0.034	99.90	81.94
DRSL[2]	$n/2$	**0.887**	**0.025**	49.70	**0.800**	**0.028**	49.69	0.805	**0.031**	49.69	**0.733**	0.033	49.76	10.49
	$n/5$	**0.879**	**0.025**	19.97	**0.786**	**0.030**	19.93	0.792	**0.031**	19.90	0.720	0.034	19.94	27.14
	$n/10$	**0.873**	**0.026**	9.98	**0.767**	**0.031**	10.01	0.752	0.032	9.95	0.699	0.034	9.98	45.58
SparsePKL/	n	0.857	0.027	99.75	0.777	0.030	99.96	0.793	**0.031**	99.88	**0.723**	0.034	99.95	1.17
DRSEI	$n/2$	0.869	0.027	49.89	0.796	0.029	49.95	**0.814**	**0.031**	49.75	0.718	0.034	49.76	3.93
	$n/5$	0.874	0.026	19.96	0.785	**0.030**	19.95	**0.799**	**0.031**	19.94	**0.721**	0.033	19.97	9.88
	$n/10$	0.840	0.028	9.97	0.761	**0.031**	9.97	**0.780**	**0.030**	9.97	**0.719**	0.033	9.97	17.33
SparsePKL/	n	0.832	0.029	99.56	0.768	0.032	99.89	0.782	0.032	99.81	0.706	0.034	99.83	**0.96**
DREIS	$n/2$	0.850	0.029	49.69	0.766	0.032	49.85	0.783	0.032	49.77	0.707	0.034	49.76	**3.29**
	$n/5$	0.839	0.029	19.95	0.767	0.031	19.91	0.784	0.032	19.92	0.710	0.034	19.96	**5.10**
	$n/10$	0.824	0.029	9.97	0.758	**0.031**	9.98	**0.780**	0.032	9.95	0.702	0.034	9.98	**8.79**
SparsePKL/	n	0.790	0.031	98.29	0.739	0.031	98.13	0.744	**0.031**	97.60	0.631	**0.032**	98.33	7.00
DREIA	$n/2$	0.805	0.031	49.79	0.743	0.031	49.76	0.742	**0.031**	49.82	0.654	**0.032**	49.78	8.21
	$n/5$	0.831	0.031	19.91	0.758	0.031	19.96	0.773	**0.031**	19.91	0.674	**0.032**	19.93	13.32
	$n/10$	0.822	0.031	9.97	0.746	**0.031**	9.96	0.754	0.031	9.94	0.651	**0.032**	9.96	16.71
SparsePKL/	n	0.748	0.032	97.07	0.651	0.032	95.45	0.610	0.032	94.69	0.541	**0.032**	95.74	11.53
DRAL	$n/2$	0.738	0.032	49.50	0.709	0.032	49.69	0.623	0.032	49.31	0.557	**0.032**	49.63	14.11
	$n/5$	0.723	0.032	19.94	0.670	0.032	19.83	0.580	0.032	19.85	0.555	**0.032**	19.85	18.80
	$n/10$	0.694	0.032	9.91	0.653	0.032	9.90	0.571	0.032	9.93	0.561	**0.032**	9.97	25.81

[1] LMBMKronℓ_0LS results are averaged over four successful runs

[2] DRSL with $k = n/10$: 1/5 infeasible solution with S2 (nz% $\approx$ 10.11)

Table 9 Results in IC

Method/	k	S1			S2			S3			S4			CPU
Objective		CI	MSE	nz%	CI	MSE	nz%	CI	MSE	nz%	CI	MSE	nz%	
CGKronRLS	n	0.945	0.020	99.98	0.810	0.024	99.96	0.742	0.035	99.84	0.598	0.036	99.95	10.60
CGKronSVM	n	0.945	0.019	99.93	0.784	0.023	99.91	0.755	0.034	99.90	0.576	0.036	99.52	85.10
LMBMKronℓ_0LS	-	0.917	0.031	13.00	0.786	0.029	48.19	0.728	0.040	10.52	0.591	0.037	27.74	325.61
SparsePKL/	n	**0.941**	**0.021**	99.99	**0.805**	**0.024**	99.97	0.745	0.035	99.95	**0.593**	0.036	99.95	191.13
DRSL[1]	$n/2$	**0.931**	**0.022**	49.74	**0.769**	**0.025**	49.68	**0.755**	0.034	49.60	**0.615**	0.035	49.59	37.31
	$n/5$	**0.928**	**0.023**	19.92	**0.755**	**0.026**	19.98	**0.752**	0.035	19.94	**0.607**	0.035	20.00	53.84
	$n/10$	**0.922**	**0.023**	10.15	**0.740**	**0.028**	9.99	0.719	0.034	10.00	0.562	0.036	10.05	130.60
SparsePKL/	n	0.914	0.025	99.69	0.749	0.027	99.80	**0.752**	0.035	99.87	0.591	0.035	99.88	2.97
DRSEI[2]	$n/2$	0.918	0.024	49.75	0.744	0.027	49.73	0.747	0.034	49.82	0.600	0.035	49.85	11.33
	$n/5$	0.915	0.024	19.93	0.733	0.027	19.95	0.743	0.034	19.97	0.588	0.035	19.98	21.29
	$n/10$	0.900	0.025	9.97	0.736	**0.028**	9.97	**0.734**	0.034	9.99	**0.578**	0.035	10.01	48.92
SparsePKL/	n	0.893	0.027	99.57	0.702	0.030	99.64	0.718	0.035	99.81	0.577	0.037	99.78	**2.84**
DREIS	$n/2$	0.908	0.026	49.61	0.724	0.029	49.68	0.732	0.035	49.79	0.579	0.036	49.66	**11.21**
	$n/5$	0.899	0.026	19.94	0.703	0.030	19.93	0.732	0.035	19.96	0.572	0.037	19.93	**14.72**
	$n/10$	0.885	0.027	9.98	0.687	0.030	9.97	0.711	0.035	9.96	0.571	0.037	9.98	**22.78**
SparsePKL/	n	0.861	0.032	98.43	0.702	0.032	98.38	0.681	**0.033**	98.20	0.544	**0.033**	98.24	22.35
DREIA	$n/2$	0.856	0.032	49.88	0.673	0.032	49.78	0.706	**0.033**	49.76	0.543	**0.033**	49.66	23.67
	$n/5$	0.844	0.032	19.94	0.676	0.032	19.93	0.669	**0.033**	19.90	0.542	**0.033**	19.94	40.91
	$n/10$	0.826	0.032	9.97	0.678	0.032	9.97	0.655	**0.033**	9.97	0.511	**0.033**	9.96	53.69
SparsePKL/	n	0.765	0.033	97.516	0.691	0.031	97.783	0.620	**0.033**	98.202	0.570	**0.033**	98.307	34.305
DRAL[3]	$n/2$	0.765	0.033	49.559	0.669	0.032	49.744	0.615	**0.033**	49.584	0.575	**0.033**	49.609	41.595
	$n/5$	0.752	0.033	19.916	0.668	0.032	19.910	0.609	**0.033**	19.861	0.570	**0.033**	19.903	67.907
	$n/10$	0.742	0.033	10.006	0.613	0.033	9.957	0.606	**0.033**	9.945	0.560	**0.033**	9.950	90.164

[1] DRSL with $k = n/10$: 3/5 infeasible solutions with S1 (nz% $<$ 10.34), 1/5 infeasible solution with S2 (nz% $\approx$ 10.04) and S3 (nz% $\approx$ 10.08), and
4/5 infeasible solutions with S4 (nz% $<$ 10.19))
[2] DRSEI with $k = n/10$: 1/5 infeasible solution with S3 (nz% $\approx$ 10.01) and S4 (nz% $\approx$ 10.04)
[3] DRAL with $k = n/10$: 1/5 infeasible solution with S1 (nz% $\approx$ 10.81)

Table 10 Results in E

Method/	k	S1			S2			S3			S4			CPU
Objective		CI	MSE	nz%	CI	MSE	nz%	CI	MSE	nz%	CI	MSE	nz%	
CGKronRLS	n	0.928	0.005	99.78	0.846	0.005	99.60	0.660	0.010	94.74	0.641	0.010	99.80	267.61
CGKronSVM	n	0.928	0.005	99.76	0.827	0.005	99.84	0.683	0.009	99.19	0.629	0.009	80.14	4467.58
LMBMKronℓ_0LS[1]	-	0.904	0.058	8.00	0.837	0.007	9.78	0.637	0.337	13.36	0.608	0.009	35.38	7956.34
SparsePKL/	n	**0.919**	**0.005**	99.67	**0.836**	**0.005**	99.59	0.703	0.009	98.08	0.597	**0.009**	99.78	2409.17
DRSL[2]	$n/2$	**0.913**	**0.006**	49.90	**0.829**	**0.005**	49.65	0.651	0.009	49.71	0.598	0.009	49.62	1048.95
	$n/5$	**0.914**	**0.006**	19.86	**0.823**	**0.006**	19.92	0.656	0.009	19.91	0.586	0.009	19.91	1070.29
	$n/10$	**0.911**	**0.006**	10.01	**0.819**	**0.006**	10.01	0.661	0.009	10.01	0.584	0.009	10.02	1313.70
SparsePKL/	n	0.874	0.008	84.75	0.787	0.007	92.87	**0.716**	**0.008**	94.15	**0.652**	**0.009**	96.61	**20.78**
DRSEI[3]	$n/2$	0.888	0.007	49.50	0.788	0.007	49.64	**0.718**	**0.008**	49.68	**0.644**	0.009	49.60	97.50
	$n/5$	0.887	0.007	19.88	0.788	0.007	19.89	**0.716**	**0.008**	19.87	**0.644**	**0.008**	19.87	148.84
	$n/10$	0.883	0.007	10.00	0.796	0.007	9.93	**0.713**	**0.008**	9.97	**0.637**	**0.008**	9.96	198.61
SparsePKL/	n	0.847	0.015	53.23	0.763	0.011	78.57	0.693	0.012	80.23	0.631	0.010	87.70	24.56
DREIS[4]	$n/2$	0.860	0.014	49.31	0.712	0.015	40.97	0.697	0.011	49.49	0.639	0.010	49.62	**87.50**
	$n/5$	0.870	0.011	19.84	0.771	0.009	19.89	0.708	0.010	19.93	0.640	0.010	19.90	**136.86**
	$n/10$	0.885	0.010	9.95	0.773	0.009	9.97	0.687	0.010	9.95	0.631	0.010	9.97	**175.44**
SparsePKL/	n	0.815	0.008	97.32	0.772	0.008	97.49	0.616	0.011	97.88	0.592	0.457	97.77	527.43
DREIA[5,6]	$n/2$	0.831	0.008	49.68	0.775	0.008	49.78	0.661	**0.008**	49.83	0.606	**0.008**	49.75	703.42
	$n/5$	0.835	0.008	19.91	0.754	0.008	19.86	0.646	**0.008**	19.94	0.609	**0.008**	19.92	996.21
	$n/10$	0.816	0.008	10.06	0.730	0.008	9.97	0.639	**0.008**	9.96	0.604	**0.008**	10.01	1108.54
SparsePKL/	n	0.656	0.008	97.41	0.617	0.011	96.62	0.581	0.009	96.11	0.553	**0.009**	96.58	820.83
DRAL[7]	$n/2$	0.655	0.008	49.68	0.613	0.009	49.73	0.568	**0.008**	49.75	0.551	0.009	49.78	1430.02
	$n/5$	0.657	0.008	19.85	0.627	0.008	19.93	0.570	**0.008**	19.87	0.553	**0.008**	19.89	1790.38
	$n/10$	0.625	0.008	10.03	0.617	0.008	10.24	0.557	**0.008**	10.09	0.553	**0.008**	10.19	2022.20

[1] LMBMKronℓ_0LS results are averaged over four successful runs
[2] DRSL with $k = n/10$: 3/5 infeasible solutions with S1 (nz% $<$ 10.07), 1/5 infeasible solution with S2 (nz% $\approx$ 10.08) and S3 (nz% $\approx$ 10.03), and 4/5 infeasible solutions with S4 (nz% $<$ 10.06)
[3] DRSEI with $k = n/10$: 2/5 infeasible solutions with S1 (nz% $<$ 10.17) and 1/5 infeasible solution with S3 (nz% $\approx$ 10.013)
[4] DREIS with $k = n/10$: 1/5 infeasible solution with S2 (nz% $\approx$ 10.03) and S4 (nz% $\approx$ 10.07)
[5] DREIA with $k = n/10$: 4/5 infeasible solutions with S1 (nz% $<$ 10.17), 1/5 infeasible solution with S2 (nz% $\approx$ 10.02) and S3 (nz% $\approx$ 10.01), and 2/5 infeasible solutions with S4 (nz% $<$ 10.14)
[6] DREIA with $k = n$: The high MSE value under setting S4 could be prevented by validating with the MSE instead of the C-index. It is due to the fact that the C-index does not care on magnitude of the values but their order
[7] DRAL with $k = n/10$: 3/5 infeasible solutions with S1 (nz% $<$ 10.10), 5/5 infeasible solutions with S2 (nz% $<$ 10.51), 3/5 infeasible solutions with S3 (nz% $\approx$ 10.26), and 4/5 infeasible solutions with S4 (nz% $<$ 10.56)

Here we only compare the results that have approximately the same sparsity levels. That is, the results of SparsePKL we use for comparison depend on the sparsity levels obtained with LMBMKronℓ_0LS. If none of the solutions is better, we use the red pen to the result obtained with LMBMKronℓ_0LS.

In the tables, $nz\%$ denotes the percentages of the variables that are nonzero at the solution, and CPU times (given in seconds) are averaged over all runs in data. That is, they include computation of all different settings. In addition, we report the number of infeasible solutions together with the biggest $nz\%$ obtained with SparsePKL as a footnote to each table.

It is worth noting that LMBMKronℓ_0LS crashed often (with an unknown reason), especially with bigger data sets. The results are averaged over the successful runs.

Funding Open Access funding provided by University of Turku (including Turku University Central Hospital). Open Access funding provided by University of Turku (including Turku University Central Hospital. This work was financially supported by University of Turku and Research Council of Finland Grants #345804 and #345805.

Data Availability All the data analysed during this study are reported in the paper and openly available.

Code availability the source code of SparsePKL is available at https://github.com/napsu/sparsePKL.

Declarations

Conflict of interest: The authors have no financial or non-financial interests to disclose.

References

1. Airola, A., Pahikkala, T.: Fast Kronecker product kernel methods via generalized Vec trick. IEEE Trans. Neural Netw. Learn. Syst. **29**, 3374–3387 (2018)
2. Bagirov, A.M., Gaudioso, M., Karmitsa, N., Mäkelä, M.M., Taheri, S.: Numerical Nonsmooth Optimization: State of the Art Algorithms. Springer, Cham (2020)
3. Bagirov, A.M., Karmitsa, N., Mäkelä, M.M.: Introduction to Nonsmooth Optimization: Theory, Practice and Software. Springer, Cham (2014)
4. Bertsimas, D., King, A., Mazumder, R.: Best subset selection via a modern optimization lens. Ann. Stat. **44**(2), 813–852 (2016)
5. Byrd, R.H., Nocedal, J., Schnabel, R.B.: Representations of quasi-Newton matrices and their use in limited memory methods. Math. Program. **63**, 129–156 (1994)
6. Cichonska, A., Pahikkala, T., Szedmak, S., Julkunen, H., Airola, A., Heinonen, M., Aittokallio, T., Rousu, J.: Learning with multiple pairwise kernels for drug bioactivity prediction. Bioinformatics **34**(13), i509–i518 (2018)
7. Cichonska, A., Ravikumar, B., Allaway, R.J., et al.: Crowdsourced mapping of unexplored target space of kinase inhibitors. Nat. Commun. **12**(3307), 146–157 (2021)
8. Clarke, F.H.: Optimization and Nonsmooth Analysis. Wiley-Interscience, New York (1983)
9. Davis, M.I., Hunt, J.P., Herrgard, S., Ciceri, P., Wodicka, L.M., Pallares, G., Hocker, M., Treiber, D.K., Zarrinkar, P.P.: Comprehensive analysis of kinase inhibitor selectivity. Nat. Biotechnol. **29**(11), 1046–1051 (2011)

10. Gaudioso, M., Giallombardo, G., Miglionico, G.: Sparse optimization via vector k-norm and DC programming with an application to feature selection for support vector machines. Comput. Optim. Appl. **86**, 745–766 (2023)
11. Gaudioso, M., Gorgone, E., Hiriart-Urruty, J.: Feature selection in SVM via polyhedral k-norm. Optim. Lett. **14**, 19–36 (2020)
12. Gönen, M., Heller, G.: Concordance probability and discriminatory power in proportional hazards regression. Biometrika **92**, 965–970 (2005)
13. Gotoh, J., Takeda, A., Tono, K.: DC formulations and algorithms for sparse optimization problems. Math. Program. Ser. B **169**, 141–176 (2018)
14. Haarala, M.: Large-scale nonsmooth optimization: variable metric bundle method with limited memory. Ph.D. thesis, University of Jyväskylä, Department of Mathematical Information Technology (2004)
15. Haarala, M., Miettinen, K., Mäkelä, M.M.: New limited memory bundle method for large-scale nonsmooth optimization. Optim. Methods Softw. **19**(6), 673–692 (2004)
16. Haarala, N., Miettinen, K., Mäkelä, M.M.: Globally convergent limited memory bundle method for large-scale nonsmooth optimization. Math. Program. **109**(1), 181–205 (2007)
17. Hager, W.W., Phan, D.T., Zhu, J.: Projection algorithms for nonconvex minimization with application to sparse principal component analysis. J. Global Optim. **65**, 657–676 (2016)
18. Halkola, A., Joki, K., Mirtti, T., Mäkelä, M.M., Aittokallio, T., Laajala, T.D.: OSCAR: Optimal subset cardinality regression using the L0-pseudonorm with applications to prognostic modelling of prostate cancer. PLoS Comput. Biol. **19**(3), e1010333 (2023)
19. Hartman, P.: On functions representable as a difference of convex functions. Pac. J. Math. **9**(3), 707–713 (1959)
20. Hertz, A., Kuflik, T., Tuval, N.: Resolving sets and integer programs for recommender systems. J. Global Optim. **81**, 153–178 (2021)
21. Hiriart-Urruty, J.B.: Generalized differentiability, duality and optimization for problems dealing with differences of convex functions. In: Ponstein, J. (ed.) Convexity Dual. Optim., vol. 256, pp. 37–70. Springer, Berlin (1985)
22. Hofmann, T., Schölkopf, B., Smola, A.: Kernel methods in machine learning. Ann. Stat. **36**(3), 1171–1220 (2008)
23. Joki, K., Bagirov, A.M., Karmitsa, N., Mäkelä, M.M.: A proximal bundle method for nonsmooth DC optimization utilizing nonconvex cutting planes. J. Global Optim. **68**(3), 501–535 (2017)
24. Kampa, K., Mehta, S., Chou, C.A., Chaovalitwongse, W.A., Grabowski, T.J.: Sparse optimization in feature selection: application in neuroimaging. J. Global Optim. **59**, 439–457 (2014)
25. Karmitsa, N., Bagirov, A.M., Mäkelä, M.M.: Comparing different nonsmooth optimization methods and software. Optim. Methods Softw. **27**(1), 131–153 (2012)
26. Kiwiel, K.C.: Methods of Descent for Nondifferentiable Optimization. Lecture Notes in Mathematics, vol. 1133. Springer, Berlin (1985)
27. Le Thi, H.A., Pham Dinh, T.: Convex analysis approach to DC programming: theory, algorithms and applications. Math. Program. **169**(1), 5–68 (2018)
28. Le Thi, H.A., Pham Dinh, T.: Open issues and recent advances in DC programming and DCA. J. Global Optim. (2023). https://doi.org/10.1007/s10898-023-01272-1
29. Liu, T.Y.: Learning to Rank for Information Retrieval. Springer, Berlin (2011)
30. Meng, N., Zhao, Y.-B., Kočvara, M., Sun, Z.: Partial gradient optimal thresholding algorithms for a class of sparse optimization problems. J. Global Optim. **84**, 393–413 (2022)
31. Menon, A., Elkan, C.: A log-linear model with latent features for dyadic prediction. In: The 10th IEEE International Conference on Data Mining (ICDM), pp. 364–373 (2010)
32. Merget, B., Turk, S., Eid, S., Rippmann, F., Fulle, S.: Profiling prediction of kinase inhibitors: toward the virtual assay. J. Med. Chem. **60**(1), 474–485 (2017)
33. Metz, J.T., Johnson, E.F., Soni, N.B., Merta, P.J., Kifle, L., Hajduk, P.J.: Navigating the kinome. Nat. Chem. Biol. **7**, 200–202 (2011)
34. Miyashiro, R., Takano, Y.: Mixed integer second-order cone programming formulations for variable selection in linear regression. Eur. J. Oper. Res. **247**(3), 721–731 (2015)
35. Natarajan, B.K.: Sparse approximate solutions to linear systems. SIAM J. Comput. **24**(2), 227–234 (1995)
36. de Oliveira, W.: The ABC of DC programming. Set-Valued Var. Anal. **28**(1), 679–706 (2022)
37. Paasivirta, P., Numminen, R., Airola, A., Karmitsa, N., Pahikkala, T.: Predicting pairwise interaction affinities with ℓ_0-penalized least squares—a nonsmooth bi-objective optimization based approach. Optim. Methods Softw. (2023, in press)
38. Pahikkala, T., Airola, A.: Rlscore: regularized least-squares learners. J. Mach. Learn. Res. **17**, 1–5 (2016)
39. Pahikkala, T., Airola, A., Pietilä, S., Shakyawar, S., Szwajda, A., Tang, J., Aittokallio, T.: Toward more realistic drug-target interaction predictions. Brief. Bioinform. **16**(2), 325–337 (2014)

40. Park, Y., Marcotte, E.: Flaws in evaluation schemes for pair-input computational predictions. Nat. Methods **9**(12), 1134–1136 (2012)
41. Pham Dinh, T., Le Thi, H.A.: DC programming and DCA: thirty years of developments. Acta Math. Vietnam **22**(1), 289–355 (1997)
42. Rendle, S., Freudenthaler, C.: Improving pairwise learning for item recommendation from implicit feedback. In: Proceedings of the 7th ACM International Conference on Web Search and Data Mining, WSDM'14, pp. 273–282. Association for Computing Machinery (2014)
43. Rockafellar, R.T.: Convex Analysis. Princeton University Press, Princeton, NJ (1970)
44. Schölkopf, B., Herbrich, R., Smola, A.J.: A generalized representer theorem. In: Proceedings of the 14th Annual Conference on Computational Learning Theory and 5th European Conference on Computational Learning Theory pp. 416–426 (2001)
45. Schrynemackers, M., Küffner, R., Geurts, P.: On protocols and measures for the validation of supervised methods for the inference of biological networks. Front. Genet. **4**, 262 (2013)
46. Tang, J., Szwajda, A., Shakyawar, S., Xu, T., Hintsanen, P., Wennerberg, K., Aittokallio, T.: Making sense of large-scale kinase inhibitor bioactivity data sets: a comparative and integrative analysis. J. Chem. Inf. Model. **54**(3), 735–743 (2014)
47. Taskar, B., Wong, M.-F., Abbeel, P., Koller, D.: Link prediction in relational data. In: Advances in Neural Information Processing Systems, vol. 16. MIT Press (2003)
48. Viljanen, M., Airola, A., Pahikkala, T.: Generalized vec trick for fast learning of pairwise kernel models. Mach. Learn. **111**, 543–573 (2022)
49. Wang, Z., Zhou, Y., Hong, L., Zou, Y., Su, H., Chen, S.: Pairwise learning for neural link prediction (2022). https://doi.org/10.48550/arXiv.2112.02936
50. Wei, Z., Li, Q., Wei, J., Bian, W.: Neural network for a class of sparse optimization with L0-regularization. Neural Netw. **151**, 211–221 (2022)
51. Yamanishi, Y., Araki, M., Gutteridge, A., Honda, W., Kanehisa, M.: Prediction of drug-target interaction networks from the integration of chemical and genomic spaces. Bioinformatics **24**(13), i232–i240 (2008)
52. Yao, Y., Rosasco, L., Caponnetto, A.: On early stopping in gradient descent learning. Constr. Approx. **26**(2), 289–315 (2007)
53. Yi, S., Lai, Z., He, Z., Cheung, Y.-M., Liu, Y.: Joint sparse principal component analysis. Pattern Recogn. **61**, 524–536 (2017)
54. Yin, Y., Wang, Y., Dai, T., Wang, L.: DOA estimation based on smoothed sparse reconstruction with time-modulated linear arrays. Signal Process. **214**, 109229 (2024)
55. Zou, H., Hastie, T., Tibshirani, R.: Sparse principal component analysis. J. Comput. Graph. Stat. **15**(2), 265–286 (2006)